Network of Excellence on
Engineering Secure Future Internet
Software Services and Systems

Network of Excellence

Deliverable D7.4

Enhanced set of solutions supporting secure service architectures and design

Project Number	:	256980
Project Title	:	NESSoS
Deliverable Type	:	Report

Deliverable Number	:	D7.4
Title of Deliverable	:	Enhanced set of solutions supporting secure service architectures and design
Nature of Deliverable	:	R
Dissemination Level	:	Public
Internal Version Number	:	
Contractual Delivery Date	:	M36 September, 30th 2013
Actual Delivery Date	:	October, 31st 2013
Contributing WPs	:	WP 7
Editor(s)	:	Benoit Baudry, INRIA
Author(s)	:	Benoit Baudry, INRIA Noel Plouzeau, INRIA Marianne Busch, LMU Manuel Clavel, IMDEA Thomas Heyman, KUL Riccardo Scandariato, KUL Kristian Beckers, UDE M. Carmen Fernandez Gago, UMA Francisco Moyano, UMA
Reviewer(s)	:	Marinella Petrocchi, CNR Christoph Sprenger, ETH Zurich Min Sang Tran, UNITN

Abstract

The main objective of work package 7 is to provide improved support for architecting and designing secure services in Future Internet applications. This report presents advances about the prototypes for secure architecture and design, which all target specific modeling challenges for trust, access control and reputation in FI applications. This spectrum reflects the different concerns that impact the security of software architecture. It also reflects the challenges that arise in a context where applications have to run in constantly evolving environments.

Keyword List

Security, Future Internet, Design, Architecture, Model-based, Decomposition, Composition, Dynamic, Adaptation, Contract, Enforcement, Reusable Know-how

Document History

Version	Type of Change	Author(s)
0.1	initial setup	Benoit Baudry, Olivier Bendavid - INRIA
0.2	initial contributions	Benoit Baudry, Noel Plouzeau - INRIA; Thomas Heyman, Riccardo Scandariato - KUL; Kristian Beckers - UDE; Carmen Fernandez Gago, Francisco Moyano - UMA
0.3	chapter about UWE extension	Marianne Busch - LMU
0.4	chapter about ActionGUI extension	Manuel Clavel - IMDEA
0.5	introduction, connections with other WPs, harmonization	Benoit Baudry - INRIA
0.6	chapter about ATOS contribution	Marina Egea - ATOS
0.7	minor fixes in all sections	all authors
0.8	update according to internal review by C. Sprenger	B. Baudry
0.81	update according to internal review by M. Petrocchi	B. Baudry
0.82	update according to internal review by M. Tran	B. Baudry
0.9	pre-final version	B. Baudry
1.0	final version	B. Baudry

Table of Contents

Acronyms

ACL	access control list
ACR	access-control reasoner
ADT	android development tools
AVD	android virtual device
CBK	Common Body of Knowledge
DAC	discretionary access control
DOM	document object model
DoS	denial-of-service attack
DoW	description of work
GWT	google web toolkit
INRIA	Institut National de Recherche en Informatique et Automatique
JDK	java development kit
JML	java modeling language
JNA	Java Native Access
JRE	java runtime environment
MAC	mandatory access control
NESSOS	Network of Excellence on Engineering Secure Future Internet Software Services and Systems
NPD	name protection domain
PAP	policy administration point
PDP	policy decision point
PEP	policy enforcement point
PIP	policy information point
POC	proof of concept
RAM	reflecting architectural model
RBAC	role based access-control
RIA	rich internet applications
RPC	remote procedure call
SAML	Security assertion markup language
SBT	self booking tool
SDK	software development kit
TAM	target architectural model
TCSEC	trust computer system evaluation criteria
UML	Unified Modeling Language
URL	Uniform Resource Locator
UWE	UML-based web engineering
WP	work package
XACML	eXtensible access-control markup language
XSRF	Cross-Site Request Forgery
XSS	Cross-Site Scripting

List of Figures

Executive summary

This deliverable focuses on methods and tools to model secure and adaptive architectures for software applications deployed on the Future Internet. It is organized around the different prototypes that have been developed by the WP partners to build secure service architectures and design. The proposed solutions are ordered according to their level of abstraction, from solutions that are the closest to requirements to solutions that are closer to the running system:

- Section 2 analyzes the ISO 27001 demands for system development and documentation with regard to cloud computing systems. Based on these insights, we provide a method that relies upon existing requirements engineering methods. Moreover, our method relies on patterns for several security tasks, e.g., context descriptions, threat analysis and policy definition. Our method can ease the effort of establishing a cloud-specific Information Security Management System (ISMS) and can produce the necessary documentation for an ISO 27001 compliant ISMS;
- Section 3 introduces a formal modeling and security verification technique that leverages model finding to verify that the security critical parts of a software architecture adhere to their security requirements. By making use of abstraction, the technique supports compositional verification and iterative refinement of these models. This, in turn, promotes reuse of (partial) verification results and enables integration in a larger software development process;
- Section 4 presents an extension of security features of UWE [28]. The aim is to be more specific about web applications (secure downloads, cross-site request forgery (CSRF) prevention, panic mode, under attack mode and so on). Another area is the transformation of graphical UWE models (in UML) to a textual DSL written in Scala. This DSL, called TextualUWE, can then be easily checked by functions written in functional Scala;
- Section 5 proposes a novel model-driven methodology, called ActionGUI, for developing secure data-management applications. In this deliverable we report on several applications that demonstrate the usefulness of our approach and the associated tools;
- Section 6 reports on a prototype that performs an automatic translation of declarative, rule-based ATL transformations into transformation models, and subsequently uses to them existing model satisfiability checkers for OCL-annotated metamodels to verify the translated ATL transformations with respect to the desired pre and postconditions. The resulting encoding of ATL into OCL-constrained models can be used for the analysis of various properties, in particular the correct mapping of security properties by transformations;
- Section 7 proposes an integration of intrusion detection capacities in a models@runtime framework. This enables detecting nodes in a distributed system that are trying to corrupt the application's model. The IDS can trigger reconfiguration to secure the application and the new model is disseminated through a secure private polling to secure the reconfiguration process itself;
- Section 8 proposes trust to be the main driver for deciding whether a self-adaptive system must change during its execution. A system can sense the environment, inspect the trust relationships among its nodes and components, and change itself as a response to changes in this information;
- Section 9 summarizes the relationships between the work developed in WP7 and the other workpackages.

1 Introduction

NESSoS WP7 focuses on methods and tools to model secure and adaptive architectures for software applications deployed on the Future Internet [19]. A major characteristic of Future Internet applications is "the fading boundary between development time and runtime" [15]. Consequently, the system architecture is designed in the early development phases, but then constantly evolves through the whole development lifecycle, including in post-deployment phases. This is reflected in the contributions integrated in WP7, which propose model-based concepts to integrate different security concerns in software architecture at in requirements definition phases, at design time and at runtime.

The workpackage contributions can thus be organized according to three dimensions: the security concern modeled and analyzed by a given technique; the phase in the lifecycle at which a given technique proposes to model and analyze security concerns; the formalism used to model security and the system. Figure 1.1 represents the different solutions the workpackage's contributions investigate in each dimension. The tools and prototypes in WP7 support security analysis according to trust, access control or security standards for information systems. These security concerns are modeled in early design models, detailed analysis models and for runtime deployment, using one or a combination of the following formalisms: UML, Secure UML, ADL (an architecture description language) or OCL.

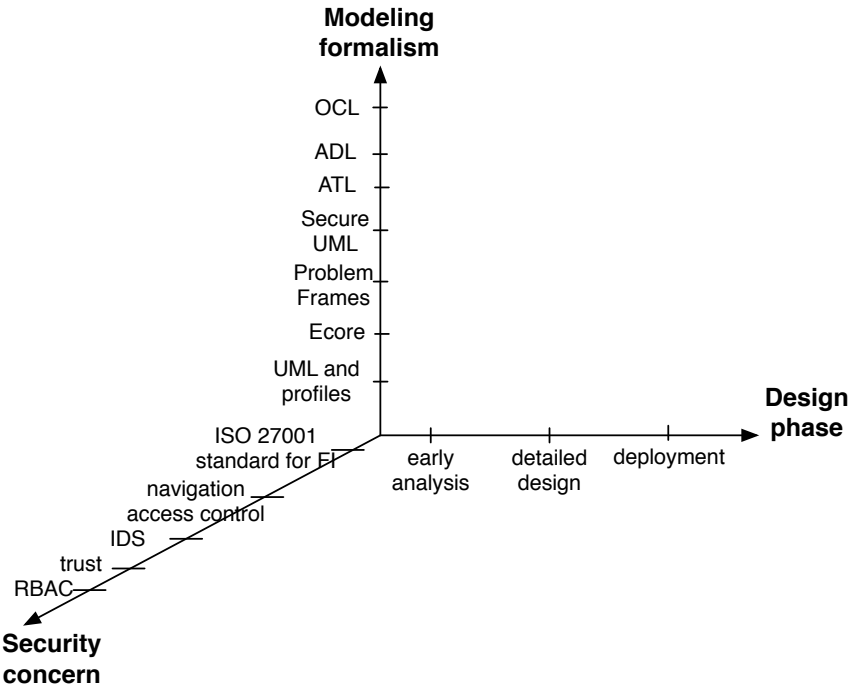

Figure 1.1: Modeling secure architectures: three dimensions explored in WP7 contributions

This deliverable focuses on the presentation of tools and prototypes that support the modeling methods and techniques integrated in WP7. The position of each prototype with respect to the dimensions of Figure 1.1 is explicitly stated at the beginning of each section, in a table that summarizes the following information:

- purpose of the prototype: this summarizes the main modeling intention that the prototype can support, as well as the main functionality the prototype proposes;
- modeling formalism: prototypes use general purpose formalisms such as UML (sometimes extended with profiles), OCL, Ecore, as well as specific languages for architecture description (ADLs) or model transformation (ATL);
- security concern: at the architecture level, the different prototypes address access control issues (in web navigation, pervasive systems and information systems), trust and reputation issues or the adaptation of standard information systems security concerns to cloud infrastructures.

1.1 Integration activities

The development of prototypes to support model-based secure software architecture, fosters the integration efforts of the network. In particular, this deliverable presents the following prototypes developed by two or more partners :

- ATOS and INRIA integrate their expertise in formal logics modeling and model transformation in a prototype that aims at verifying the preservation of security properties through automatic model transformation (Section 6);
- UDE and SINTEF collaborated on the PACTS method described in Section 2, to integrate the CORAS risk analysis method;
- IMDEA and ETH have integrated their expertise in formal modeling and security to develop a toolbox for the automatic generation of secure GUIs (Section 5);
- UMA and INRIA have integrated the trust and reputation framework developed at UMA into the Kevoree framework to support trust-based reasoning for runtime reconfiguration (Section 8).

2 A pattern-based method for establishing a cloud-specific information security management system

Purpose of the prototype	ISO 27001 realization for cloud computing systems
Modeling formalism	UML-based
Security concern	Cloud-specific security management compliant to ISO 27001
Partner(s)	UDE

This chapter presents a summary of our publication in Springer's Requirements Engineering Journal at the special issue for security, privacy and services in the cloud [3]. In this work, we developed a method called *PACTS* that supports the cloud-specific establishment of an information security management system for cloud computing systems. The work is the continuation of our contribution to D7.3.

2.1 Introduction

The possibility of quickly acquiring or disposing of resources such as storage and memory provides a great attraction for a variety of customers. Cloud computing systems (or simply clouds) provide the means for this kind of acquisition or deposition. However, potential customers are still reserved when it comes to using cloud resources. In 2009, a study was conducted by the International Data Corporation [1] about this issue. It pointed out that security is a significant barrier for the acceptance of clouds in companies. The lack of trust in cloud security lies within the nature of clouds: storing and managing critical data and executing sensitive IT-processes is performed beyond the company's/customer's control. To gain the customer's trust and to illustrate that security is taken seriously, cloud providers have to certify their services with respect to security. One way of doing that is to turn to standards that put security at the center of interest. Several well-known companies have adopted this approach such as Microsoft[2], Amazon [3], Google [4][5], and Salesforce [6]. The aim of the ISO 27001 standard is to establish an Information Security Management System (ISMS). To use this standard for cloud computing systems is in accordance with the German Federal Office for Information Security (BSI)[7]. The current version of the standard does not take cloud specific security issues into consideration. The BSI recommends to consider cloud specific threats when dealing with cloud systems. The Cloud Security Alliance [8] and Gartner [14] have identified several of these threats. We take their findings and use them in our work. Assembling an ISMS according to the ISO 27001 standard is non-trivial task. This is supported by the fact that descriptions for system development and documentation are rather sparse. For example, the required input for the *scope and boundaries* description is to consider "characteristics of the business, the organization, its location, assets and technology" [21, p. 4]. No further information beyond that is given.

We present our *PAttern-based method for establishing a Cloud-specific informaTion Security management system (PACTS)*. We analyzed the activities demanded by the standard to build an ISMS and present patterns for these incorporating existing security requirements approaches, where applicable. We also provide a structured method that shows how the different elements described above have to be applied in order to create the required ISMS documentation admissible for certification. We use existing research on context descriptions for clouds in our method in order to provide a domain-specific approach. The patterns define stakeholders and technological artifacts that are used in the context description and all subsequent patterns and models, e.g., security policies. Furthermore, we provide relations from these patterns to cloud-specific lists of threats proposed by the Cloud Security Alliance (CSA) [8] and Gartner [14]. In addition, our approach provides a structured refinement of the cloud system's and stakeholder's

[1] https://www-304.ibm.com/isv/library/pdfs/cloud_idc.pdf
[2] http://www.windowsazure.com/en-us/support/trust-center/compliance/
[3] http://aws.amazon.com/security/
[4] http://googleenterprise.blogspot.com.br/2012/05/google-apps-receives-iso-27001.html
[5] http://www.computerweekly.com/news/2240150882/Google-Apps-for-Business-wins-ISO-27001-certification
[6] http://www.salesforce.com/platform/cloud-infrastructure/security.jsp
[7] https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Minimum_information/SecurityRecommendationsCloudComputingProviders.pdf

information to assess the threats for a particular instantiation of our cloud pattern. Our method uses this information for risk assessment and security control selection according to the ISO 27001 standard. Moreover, the ISO 27001 standard demands consideration of privacy and legal compliance. We integrated existing pattern-based research for compliance and privacy requirements into our approach in order to satisfy these demands.

2.2 Background

We illustrate our cloud system analysis pattern in Sect. 2.2.1 and we introduce the 27001 standard in particular in Sect. 2.2.2.

2.2.1 Cloud System Analysis Pattern

We propose patterns for a structured domain knowledge elicitation. Depending on the kind of domain knowledge that we have to elicit for a software engineering process, we always have certain elements that require consideration. For this work we use a specific *context elicitation pattern*, the so-called *cloud system analysis pattern* [4]. We base our approach on Jackson's work on Problem Frames [23] that con-

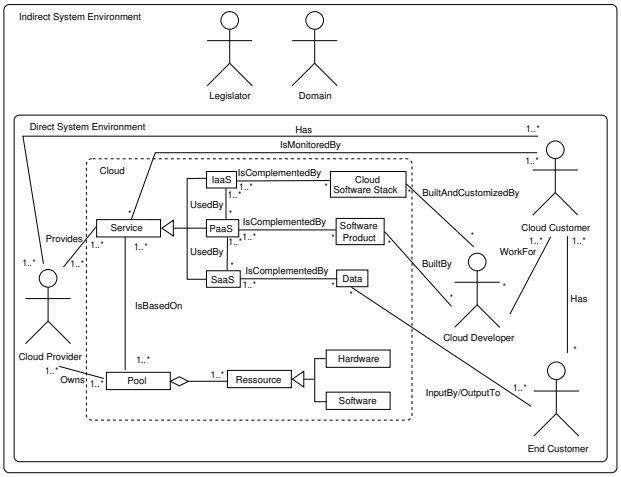

Figure 2.1: Cloud system analysis pattern taken from [4]

siders requirements engineering from the point of view of a machine in its environment: The machine is the software to be built and requirements specify the effect the machine is supposed to have on the environment. Any given environment considers certain elements, e.g., stakeholders or technical elements. Jackson [23], who describes Problem Frames as follows: *"A problem frame is a kind of pattern. It defines an intuitively identifiable problem class in terms of its context and the characteristics of its domains, interfaces and requirement."*. We were also inspired by Fowler [11], who developed patterns for the analysis phase of a given software engineering process. His patterns describe organizational structures and processes, e.g., accounting, planning, and trading.

Our patterns for the analysis phase differ from patterns concerning solutions for the design phase of software engineering like the Gang of Four patterns [13] or the security patterns by Schumacher et al. [39]. The reason is that we provide a means for a structured elicitation of domain knowledge for cloud computing systems. We do not provide solutions for the implementation phase of clouds.

We present a short introduction of our so-called *Cloud System Analysis Pattern (or short: Cloud Pattern)* [4] in the following. We created the pattern for cloud-specific context establishment and asset identification compliant to the ISO 27000 series of standards. A *Cloud* (see Fig. 2.1) is embedded into an environment consisting of two parts, namely the *Direct System Environment* and the *Indirect System Environment*. The *Direct System Environment* contains stakeholders and other systems that directly interact with the *Cloud*, i.e. they are connected to the cloud by associations. Moreover, associations between stakeholders in the *Direct* and *Indirect System Environment* exist, but not between stakeholders in the

Indirect System Environment and the *Cloud*. Typically, the *Indirect System Environment* is a significant source for compliance requirements. The *Cloud Provider* owns a *Pool* consisting of *Resources*, which are divided into *Hardware* and *Software* resources. The provider offers its resources as *Services*, i.e. *IaaS*, *PaaS*, or *SaaS*. The boxes *Pool* and *Service* in Fig. 2.1 are cloud concepts and it is not necessary to instantiate them. Instead, the specialized cloud services such as *IaaS*, *PaaS*, and *SaaS* and specialized *Resources* are instantiated. The *Cloud Developer* represents a software developer assigned by the *Cloud Customer*. The developer prepares and maintains an *IaaS* or *PaaS* offer. The *IaaS* offer is a virtualized hardware, in some cases it is equipped with a basic operating system. The *Cloud Developer* deploys a set of software named *Cloud Software Stack* (e.g. web servers, applications, databases) into the *IaaS* in order to offer the functionality required to build a *PaaS*. In our pattern *PaaS* consists of an *IaaS*, a *Cloud Software Stack* and a *cloud programming interface (CPI)*, which we subsume as *Software Product*. The *Cloud Customer* hires a *Cloud Developer* to prepare and create *SaaS* offers based on the CPI, finally used by the *End Customers*. *SaaS* processes and stores *Data* input and output from the *End Customers*. The *Cloud Provider*, *Cloud Customer*, *Cloud Developer*, and *End Customer* are part of the *Direct System Environment*. Hence, we categorize them as *direct stakeholders*. The *Legislator* and the *Domain* (and possibly other stakeholders) are part of the *Indirect System Environment*. Therefore, we categorize them as *indirect stakeholders*. We also provide templates for each stakeholder that describe their attributes in detail.

2.2.2 The ISO 27001 Standard

The ISO 27001 defines the requirements for establishing and maintaining an ISMS [21]. In particular, the standard describes the process of creating a model of the entire business risks of a given organization and specific requirements for the implementation of security controls. The ISO 27001 standard is structured according to the “Plan-Do-Check-Act” (PDCA) model, the so-called *ISO 27001 process* [21]. In the *Plan* phase an ISMS is established, in the *Do* phase the ISMS is implemented and operated, in the *Check* phase the ISMS is monitored and reviewed, and in the *Act* phase the ISMS is maintained and improved. In the *Plan* phase, the *scope and boundaries* of the ISMS, its *interested parties*, *environment*, *assets*, and all the *technology* involved are defined. In this phase also the ISMS *policies*, *risk assessments*, *evaluations*, and *controls* are defined. Controls in the ISO 27001 are measures to *modify risk*. The ISO 27001 standard demands the creation of a set of documents and the certification of an ISO 27001 compliant ISMS is based upon these documents.

Changes in the organization or technology also have to comply with the documented ISMS requirements. Furthermore, the standard requires periodic audits to ensure the effectiveness of an ISMS. These audits are also conducted using documented ISMS requirements. In addition, the ISO 27001 standard demands that management decisions, providing support for establishing and maintaining an ISMS, are documented as well. This support has to be documented via management decisions. This has to be proven as part of a detailed documentation of how each decision was reached and how many resources are committed to implement this decision.

2.3 Overview of our PACTS Method

Evaluating business benefits against privacy, security, and compliance concerns of clouds is difficult, because implementation and operational details are often not transparent to cloud customers or end customers. These stakeholders entrust their data to a cloud provider, which leads to concerns regarding data integrity, recovery and location, as well as legal issues [14].

We address these concerns by proposing our PACTS method for creating a cloud-specific ISMS compliant to the ISO 27001 standard with a particular focus on legal compliance and privacy. PACTS considers either the *cloud provider* or the *cloud customer* as possible stakeholders, who build an ISMS. The reason is that these are organizations that should earn the trust of their customers via certifying an ISMS.

Our *cloud system analysis pattern* (see Sect. 2.2.1) provides a basic structure of a cloud computing architecture, which considers the relations between stakeholders and the cloud. The pattern can be instantiated for any given cloud scenario and, if required, can be extended with little effort. The pattern provides a basis for cloud-specific asset identification, threat analysis, risk management, and control

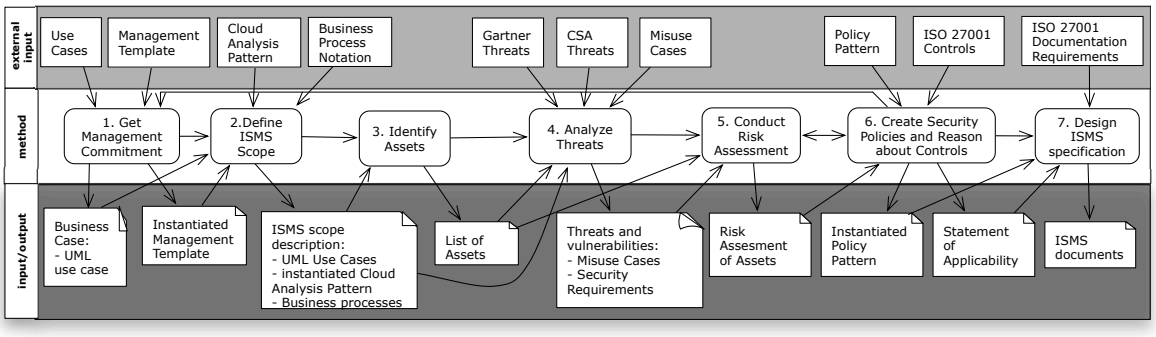

Figure 2.2: The steps of our PACTS method concerning security

selection. For example, several threats are already mapped to the cloud pattern and can be analyzed based upon the patterns instantiation. The instantiated pattern is also the input for our identification of relevant laws and analysis of privacy requirements.

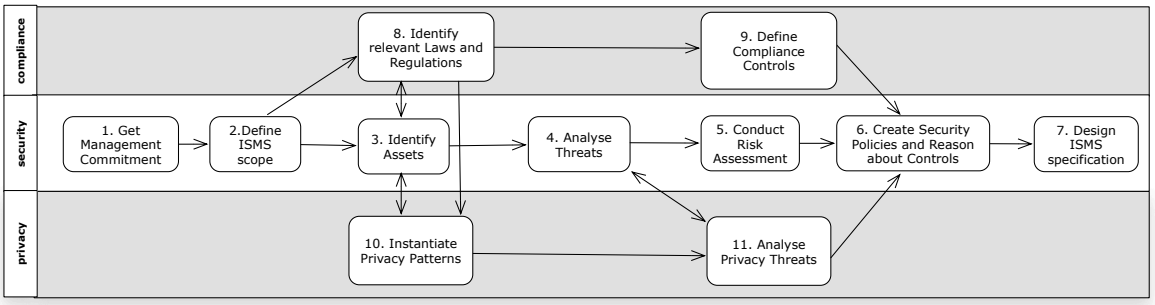

Figure 2.3: The steps of PACTS concerning compliance and privacy

Our cloud pattern reduces the effort for creating a description of a cloud. We can simply instantiate the pattern in order to get a description. The benefit of basing our method on the cloud pattern is also that knowledge collected using the pattern can be re-used for different instantiations of the pattern. For example, assets identified using the pattern can be instantiated for different projects, e.g., the *Data* in the cloud pattern has been identified as an asset. Hence, all instantiations of *Data* are assets as well. In additions, experiences from using our method can also lead to an improved pattern, e.g., the pattern can be extended with further stakeholders.

We present an overview of our method for establishing a cloud-specific ISMS in this section. In the remainder of the section we provide detailed descriptions of each step of our PACTS method. We begin by describing the steps concerning security, depicted in Fig. 2.2.

Step 1: Get Management Commitment - The precondition for building an ISMS is that the management commits to it. Thus, we dedicate the first step of our method to get management commitment for the ISMS and the provision of adequate resources to establish it. We describe the characteristics of the business via UML use case diagrams [42]. The use cases are accompanied by our management templates, which have to be instantiated with relevant information for building the ISMS, e.g., high level security goals, cloud-specific management concerns, and resource management.

Step 2: Define ISMS scope - The scope for building the ISMS shall be described using the initial use cases. These are refined using our cloud system analysis pattern for structural description of the cloud

scenario and a business process notation for behavioral description. In our examples, we choose UML activity diagrams [42] as business process notations.

Step 3: Identify Assets - The entire ISMS scope description is the input for the asset identification. We identify all items of value to the cloud stakeholders by iterating over the relations from cloud stakeholders to cloud elements in the cloud system analysis pattern and activity diagrams. This results in a list of assets and the stakeholders that own them as an output of this step of the method.

Step 4: Analyze Threats - We conduct a threat analysis using the list of threats released by the Cloud Security Alliance (CSA) [8], an industrial consortium that investigated practical security issues with clouds and the threats that Gartner [14] considers. We propose to identify threats to the previously identified assets using our cloud system analysis pattern. This activity includes an investigation of vulnerabilities of cloud components. We use the identified threats as an input for misuse cases. The results of the misuse cases are specific threats and security requirements.

Step 5: Conduct Risk Assessment - The assets, threats, vulnerabilities, and security requirements serve as input for our risk assessment. We conduct an asset-based method that uses the previously elicited knowledge to derive likelihood and consequences scales, as well as acceptable risk levels. This information is used to determine, which cloud threats cause unacceptable risks.

Step 6: Create Security Policies and Reason about Controls - Controls in the ISO 270001 standard reduce risks to assets. The reasoning about controls considers the risks to each assets and supports the decision if a control is needed or not. For each asset, we propose to compile a list that states why a control in the normative ANNEX A of the ISO 27001 should or should not be applied to that asset. We instantiate our policy pattern to ease this activity. The policy patterns help to define precisely which elements of the cloud pattern the control refers to and the security goal the control shall achieve. If the decision is made that a control has to be introduced, we go back to the previous step of our method in order to adjust the risk assessment for that particular asset. This information is in turn used to check if the control already results in an acceptable risk level or if it has to be modified or another control should be introduced. The resulting information is used to compile the so-called *Statement of Applicability*, which is a mandatory document for reasoning about the ISO 27001 controls.

We also have to check carefully if our use cases defined in Step 1 can lead to acceptable risk levels using reasonable and affordable controls. Hence, we also have to consider changing the use cases in case acceptable risk levels cannot be achieved with reasonable efforts.

Step 7: Design ISMS specification - The final step of our method concerns the ISO 27001 specification, an implementable description of the ISMS. We consider the ISO 27001 documentation demands and use the information elicited and documented in the previous steps of our method. This information is mapped to the required document types. These documents are also the basis for a certification of an ISO 27001 compliant ISMS.

The ISO 27001 standard has demands for quality requirements beyond security, namely compliance and privacy. We provide support for eliciting and analyzing these requirements as part of our approach. We show how compliance and privacy concerns are addressed in Fig. 2.3. The consideration of compliance issues for cloud computing systems is also a key recommendation of Gartner's [14] analysis.

In our PACTS method compliance identifies relevant laws and regulation and defines corresponding requirements. The relation between compliance and privacy is that the compliance part identifies relevant laws that concern privacy. The privacy part of our method uses these laws as input.

Step 8: Identify relevant Laws and Regulations - We use the information from the asset identification. Namely we identify relevant laws and regulations with this information. This activity also has to identify assets in terms of laws and regulations, which can be related to assets in terms of security. Moreover, laws and regulations can regulate privacy concerns. This information is used during the *instantiate privacy patterns* step of our method.

Step 9: Define Compliance Controls - Once laws and regulations are identified, they have to be translated into ISO 27001 compliance controls. This translation is difficult, because in some cases laws or regulations demand reasoning about a specific concern or they demand a specific functionality. In addition, compliance controls can have relations to other ISO 27001 controls. For example, a law could demand a specific control in a certain situation, while the risk assessment results would not.

The ISO 27001 standard demands also the consideration of privacy in the informative ANNEX B. We propose the following steps to address this concern.

Step 10: Instantiate Privacy Patterns - We use textual privacy patterns based upon the ISMS scope definition and relevant laws and regulations. These patterns can be instantiated and they give rise to initial privacy requirements. In addition, the identified assets for security can be considered, because if these contain personal information they can also support instantiating further privacy patterns.

Step 11: Analyze Privacy Threats - We use a privacy threat analysis based on information flow between requirements. We analyze the flow of personal information based on the previously instantiated privacy patterns and functional requirements of the cloud scenario. We also refine the initial privacy requirements. The information flow among the requirements shows, which stakeholders have potentially access to which personal information. Afterwards, software engineers have to check if the requirements have to be modified in order to be privacy preserving.

2.4 Conclusions

The decision whether a cloud service is chosen by a costumer relies, amongst other reasons, on how trustworthy the cloud system is. One way to establish this trust is to demonstrate that security, privacy, and compliance are taken seriously by the cloud provider. This is usually achieved by providing certified services. A well-known standard for such a certification is the ISO 27001 standard. However, establishing an ISMS as required by this standard is a non-trivial task. Furthermore, the standard does not yet take the special needs of cloud computing into consideration. With the work presented in this chapter we intend to close the aforementioned gap. We do so by providing a structured pattern-based method to establish an Information Security Management System (ISMS) according to the ISO 27001 standard. It has been tailored to suit the demands for the cloud computing domain. We introduce specific patterns for clouds to elicit the context of the envisioned ISMS. The approach further allows to refine the initially elicited context with behavior descriptions. It also provides the means for documenting management commitment, threat and risk analysis, as well as a pattern-based definition of security policies compliant to the ISO 27001 standard. We enhance the approach by providing validation conditions that can be used to check the instantiated context as well as policy patterns. It is, for example, possible to check whether a given responsible stakeholder in the policy pattern is also present in the context pattern. Moreover, we take the standard's demand to consider legal compliance and privacy into account.

In summary, the benefits of our approach are:

- A structured method for establishing a cloud-specific ISMS compliant to ISO 27001;
- Detailed steps for asset identification, threat analysis, risk management and security reasoning;
- The pattern-based method provides the means for consistency checks e.g. for the instantiation of the pattern;
- Consideration of legal compliance via steps for identifying laws and regulations;
- Support for formulating and validating privacy requirements and conducting a privacy threat analysis;
- A systematic support to generate the required ISMS documentation in compliance to the standard;
- Integration of proven existing methods e.g. CORAS and Misuse Cases;

- Integrating requirements engineering for security, legal compliance, and privacy to construct a holistic ISMS.

In the future, we plan to provide a UML model for the cloud and policy patterns and implement the consistency checks using the Object Constraint Language (OCL) [43]. We currently work on providing tool-support for generating documents from our instantiated patterns in accordance with the ISO 27001 standard.

3 Compositional verification of secure software architectures

Purpose of the prototype	verification of security properties on architecture models
Modeling formalism	logic and architecture description language
Security concern	secure architecture invariants
Partner(s)	KUL

This chapter summarizes the work presented in [16]. Formal modeling is becoming a mainstream research field in software engineering and is permeating all facets of software development. Promising work has been carried out to apply formal methods to early models such as requirement models [45] and secure design [26]. The benefits of formal techniques are apparent: a formal model is the scaffolding for the precise reasoning about system qualities. However, formal methods are often criticized because of two concerns: their steep learning curve, and their limited ability to scale up to realistically sized systems.

We introduce a formal modeling technique for secure software architectures [17]. That work leverages model finding and the Alloy formal modeling language [22] as a means to verify that the security critical parts of a software architecture adhere to their intended requirements. Additionally, the technique embraces the idea of iterative refinement, where the verification process carries on continuously from the early, possibly partial model of the architecture down to the finished architectural design. The approach tackles the first concern (usability) by showing that it is possible to package catalogs of reusable formal models for commonly used design building blocks—specifically, security patterns [18], as patterns are commonly used as means of refinement in architectural models. That allows architects to transparently build more complex formal models by selecting and composing the elementary models from catalogs.

However, the resulting models (which we call 'monolithic') easily grow beyond the size that can be handled by state-of-the-art verification tools, including the Alloy analyzer [1]. Furthermore, the level of detail of the uncovered weaknesses (i.e., the quality of the analysis) is directly proportional to the level of detail with which the architecture is modelled. In general, these tools do not manage to efficiently verify models with sufficient level of detail.

In practice, this problem is solved by simplification via abstraction, in which fine-grained implementation details are omitted. This allows compositional security properties to be verified on an abstract composition, and detailed verification of individual components on separate, refined component models. However, care must be taken to ensure that the decomposition is correct. This largely boils down to two specific problems: First, as abstractions are necessarily simpler than refinements, we need to be certain that compositional properties still hold when we swap out an abstraction for its refinement. Second, we require guarantees that a refinement does not exhibit behaviour that would introduce conflicts on a compositional level.

Our key contribution is a theoretical result that can be leveraged to overcome these scalability issues. That result (in the form of three theorems) provides the necessary guidance on how to decompose an architectural model into parts in a way that (1) properties on these parts can be independently verified, (2) the parts can be replaced with more abstract representations, and hence (3) overall properties on the complete model (which we call 'modular') can be verified at a fraction of the computational cost that would be necessary for the monolithic model. Additionally, a much deeper level of analysis can be achieved, meaning that more sophisticated issues can be uncovered. Finally, we provide a road map that shows how to use tools such as the Alloy Analyzer to actually implement the proposed theoretical framework.

3.1 Summary of the approach

In order to make an abstraction from a part of a large detailed model description, the principle of information hiding and encapsulation is used. Essentially, a component can be represented at different levels of abstraction. A high level of abstraction (i.e., an *abstract* model description) allows the modeller to reason efficiently about properties on the composition level, incorporating the abstract component in a model of a larger system, and verifying what guarantees the component needs to fulfil in order for that composition

[1] http://alloy.mit.edu/alloy/

to be secure. On the other hand, a low level of abstraction (i.e., a *refined* model description) allows the modeller to model and verify the inner workings of that component up to an arbitrary level of detail. The abstract component can be refined to a more detailed component specification, or even decomposed in a proper sub-architecture, until the modeller is satisfied that the detailed model description is both secure and implementable.

Ideally, when the abstract and refined component descriptions share the same public interface, i.e., the operations that are relevant to the composition, we want to be able to transparently replace the refinement with an abstraction in that composition and obtain the same results. As security properties of interest we consider security requirements, which are operationalized security goals that are assigned to specific actors. In our work, a security requirement is a temporal logic formula expressed over the interface of a component. By requiring that abstract and refined components share that same interface, security requirements are equally applicable to both. This allows us far more flexibility in modeling and verifying software architectures, as shown in Figure 3.1: Instead of simply verifying one monolithic model, we can split up a large model by isolating individual components. Every component can be verified in isolation and subsequently refined to analyze local properties, and collected in reusable libraries of already verified model parts, while global properties are verified on an abstract composition.

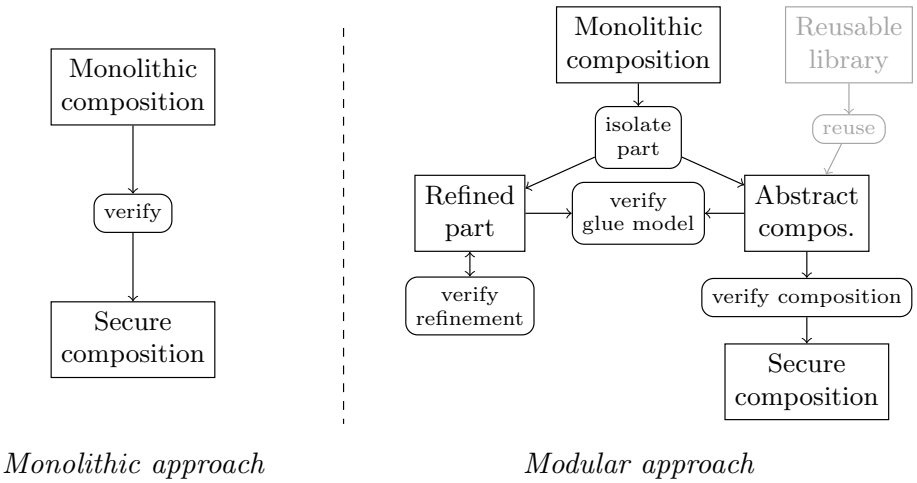

Figure 3.1: A comparison of the monolithic versus the modular verification approach.

Our modular verification technique is founded on the following practices. We have created a simple architectural meta model that is still sufficiently expressive to model software architectures by taking into account information from logical, physical and scenario views. The metamodel is shown in Figure 3.2. We have encoded that metamodel in the Alloy formal modeling language. This facilitates creating formal Alloy models of software architectures, as the metamodel connects informal software architecture descriptions (expressed, for instance, in UML) to our formal specification. More details of the modeling and verification technique can be found in [17].

As an illustration, consider an example software architecture as shown in Figure 3.3: A company has deployed a service, and needs to audit all invocations made to that service. To achieve this, the software architect introduces an interception facility (called the *Audit System*) to intercept all service invocations and generate audit events which are written to a log. The logging functionality is implemented by a *Logger* component. Furthermore, as we only want authorized users to access the service, the service contains authorization functionality, realized by the *Authorization Enforcer* component. We are interested to verify the requirement that for every execution of the operation 'ProcessRequest' on the 'ServiceIF' interface, the request is authorized and a corresponding audit event is stored by the audit system. This requirement is readily encoded as a temporal logic predicate over the public interface of the system as the Service component offers the ProcessRequest operation, the AuthorizationEnforcer allows to check whether an invocation of that operation is authorized, and the AuditSystem exposes an operation to read back the audit log. The security requirement then becomes that *for all calling components, services and*

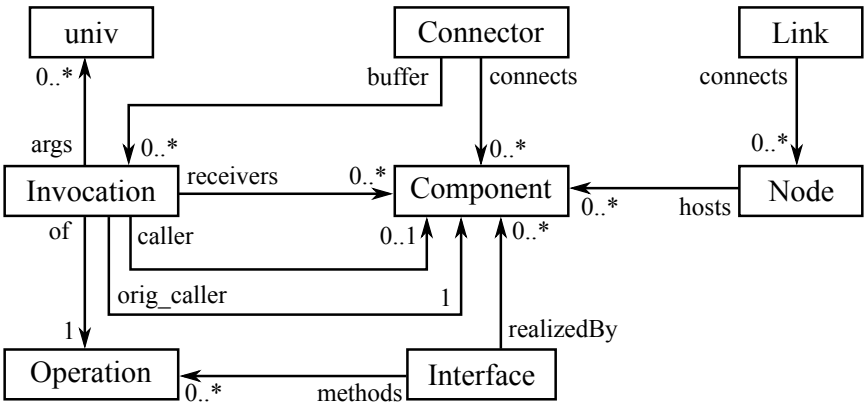

Figure 3.2: Our architectural metamodel (using UML syntax).

requests: it is always so that if the ProcessRequest operation is executed on that request, then the calling component is authorized to access that service and there exists an AuditInterceptor so that in the future that request can always be read back from that AuditInterceptor.

Verifying that requirement on a naive, monolithic Alloy model does not scale well, as can be seen in the monolithic overhead data in Figure 3.4. For larger scopes (*i.e.*, a more in-depth analysis, over models that contain more elements), the verification easily takes hours. On the other hand, if we decompose the monolithic Alloy model into an abstract composition containing an abstract Service and AuditSystem, which are in turn refined into separate models containing a detailed AuthorizationEnforcer, resp. AuditInterceptor and Logger, we can verify compositional properties such as the example security requirement in a fraction of the cost, as indicated by the composition data in Figure 3.4.

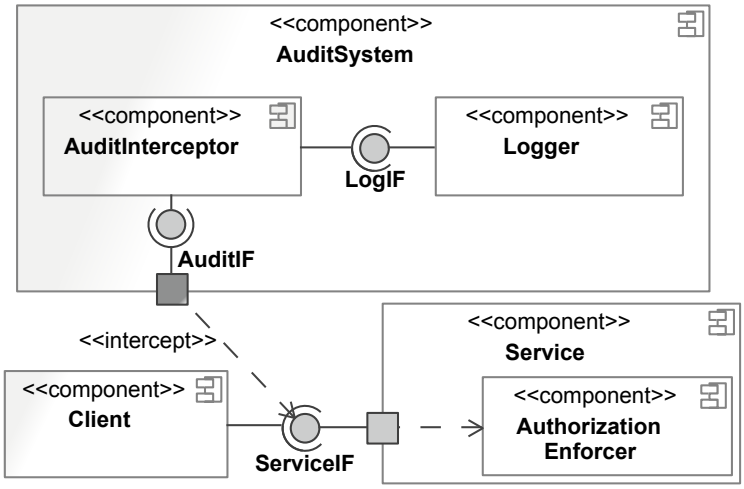

Figure 3.3: Logical view on an example architecture, providing access control and auditing functionality.

We decompose this monolithic model by isolating parts of the architecture (i.e., one or more components) that are replaced by one abstract component. We then verify that the abstraction and original, refined part are behaviorally equivalent. This entails that the pre- and postconditions of the operations that are shared by the abstraction and refinement are logically equivalent. This verification is facilitated by the creation of a so-called glue model. Not only does this glue model allow us to verify behavioral equivalence with the Alloy Analyzer, it also supports introducing systematic simplifications in the form of correspondence assumptions. These assumptions allow the software architect to hide internal implementation details while maintaining full traceability of the exact simplifications under which the abstraction

and refinement would not behave equivalently. It is then up to the software architect to accept these correspondence assumptions, or fine-tune either the abstraction or refinement.

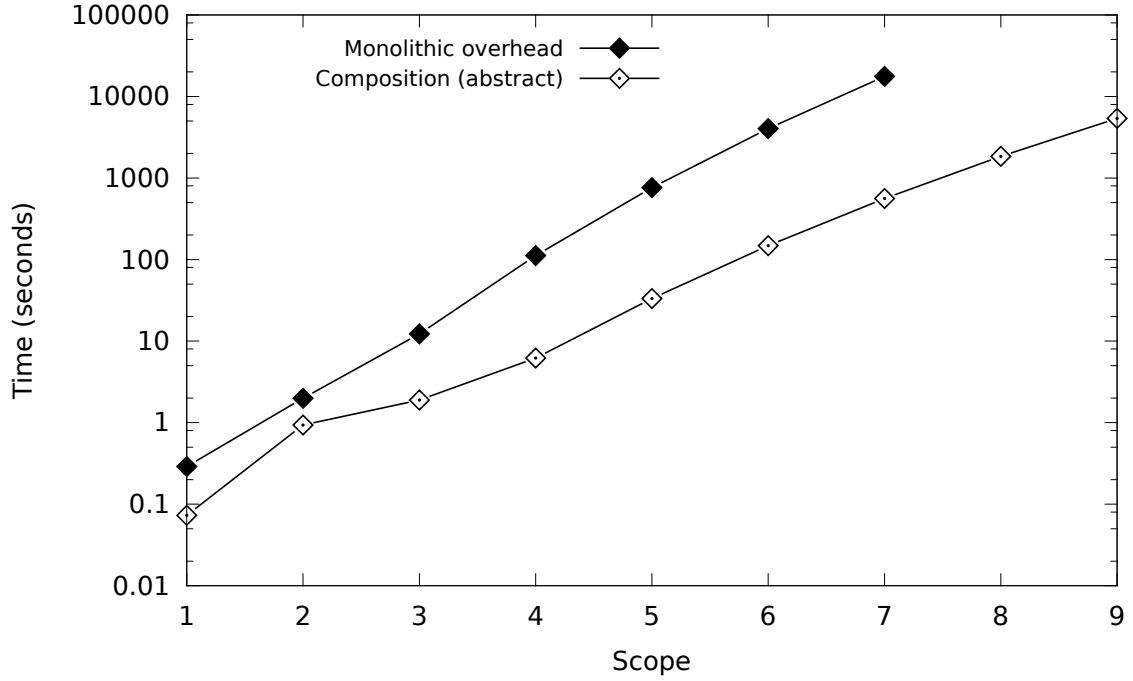

Figure 3.4: Verification performance of the composition in function of the scope, for both the modular and monolithic model descriptions

Finally, we leverage first order logic and model theory to prove that a (refined) part of a software architecture can be replaced with a behaviorally equivalent abstraction. This entails the following two things. First, if a property is shown to hold on the abstract composition, then it equally holds over the original refined composition. Second, if the abstract composition still has instances (i.e., it does not introduce conflicts), then the original refined composition also has instances.

3.2 Results

Verifying global properties on an abstract composition is an order of magnitude faster than verifying the same property on the original refined composition, as shown in Figure 3.4. Additionally, the refined parts can now iteratively be refined down to an arbitrary level of detail, and be decomposed in turn when they too become too unwieldy. The modular approach is compared to the monolithic approach in Figure 3.1.

Additionally, and perhaps most importantly, the modular approach allows us to create reusable libraries of already verified models, such as security patterns [18]. This allows the software architect to efficiently create compositions of these other models, which can be independently verified to uphold certain properties (e.g., fine-grained security requirements) by experts. Finally, modular verification brings our formal modeling and verification process closer to industrial practice, as they remove the technical limitations of automated formal verification to scale up to realistically sized systems, and significantly help in reducing the learning curve to create architectural compositions that contain sufficient detail to generate non-trivial results.

4 A UML extension for modeling protection-specific security features for web applications

Purpose of the prototype	*Specify RBAC and protection-specific behavior in web application models*
Modeling formalism	*UWE Models*
Security concern	*RBAC, navigational access control and web-related security features*
Partner(s)	*LMU*

To secure web applications is increasingly important because of rising cybercrime as well as the growing awareness of data privacy. Besides confidential connections and authentication, both data access control and navigational access control are the most relevant security features in this field. However, adding such security features to already implemented web applications is an error-prone task. Therefore, the goal is to include security features in early stages of the development process, i.e., at requirements specification and design modeling level.

Existing approaches, such as OOHRIA [29], OOWS [44], WebML [30], UWE [27, 6], or ActionGUI [2] already provide well-known methods and tools for the design of web applications. Most of them follow the principle of "separation of concerns" using separate models for different views on the application, such as e.g. content, navigation, presentation and business processes. Most of the available methods do not support modeling security-features, whereas the UWE approach by Koch et al. [6] and the ActionGUI approach by Basin et al. [2] define models for security features as access control and authorization. ActionGUI's proprietary notation comprises data models, security models and GUI models, and the application logic is represented using OCL. UWE provides a set of UML stereotypes for each view, defined by a so-called UML profile. UWE's main focus is on the process of discussing and planning an application from different points of view as e.g. requirements, content (data model), navigation, users and roles, basic access control rights, presentation and process.

ActionGUI and UWE have been developed for many years, trying to abstract from as many implementational details as possible. What seems to be the right way for most modeling methods, raises questions when dealing with secure web applications: How should Cross-Site-Request-Forgery (CSRF[1]) be prevented? What should happen in case the web application is under attack, e.g., under denial-of-service (DoS[2]) attack? So far, those questions tend to be answered in lengthy specification documents or they are just documented by the code itself. Neither approach contribute to a quick understanding of the way how web-related security concerns are managed for a certain web application.

Our approach is to address the answer to those questions at design level extending the set of modeling elements provided by UWE in order to be able to express protection-specific security concerns. This means we extend UWE's UML profile to support modeling the solutions deployed to shield a web application and its users against attacks. Therefore, we find means to express security-related solutions for the web, even though we maintain the necessary abstraction a modeling language needs.

In the following, we outline UML-based Web Engineering (UWE) [6], the security-aware engineering approach we have chosen for modeling web applications. Afterwards, we describe our extension for solution-specific security features, which is work in progress.

4.1 UML-based web engineering – UWE

One of the cornerstones of the UWE language is the "separation of concerns" principle using separate models for different views. However, we can observe that security features are cross-cutting concerns which cannot be separated completely. The main UWE models are:

The Requirements Model defines (security) requirements for a project.

[1] CSRF exploits the trust that a web site has in a user's browser (and its cookies) when a user is logged in. For example, the user clicks on an URL in an email and the site executes an action which was encoded in the URL, as signed in user.

[2] A DoS attack is characterized by a huge number of parallel requests, which are initiated by attackers, so that legitimate requests cannot be answered any more.

The Content Model contains the data structure used by the application.

The UWE Role Model describes a hierarchy of user groups to be used for authorization and access control issues. It is usually part of a *User Model*, which specifies basic structures, as e.g., that a user can take on certain roles simultaneously.

The Basic Rights Model describes access control policies. It constrains elements from the *Content Model* and from the *Role Model*.

The Presentation Model sketches the web application's user interface.

The Navigation State Model defines the navigation flow of the application and navigational access control policies. The former shows which possibilities of navigation exist in a certain context. The latter specifies which roles are allowed to navigate to a specific state and the action taken in case access cannot be granted. In a web application such actions can be, e.g., to logout the user and to redirect to the login form or just to display an error message. Furthermore, secure connections are modeled.

For each view, an appropriate type of UML diagram is used, e.g., a state machine for the Navigational Model. In addition, the UWE Profile adds a set of stereotypes, tag definitions and constraints, which can be downloaded from the UWE website [28]. Stereotypes can then be applied to UML model elements and values can be assigned to tags, which are associated to at least one stereotype.

4.2 Modeling protection-specific security features with UWE

To model protection-specific features with UWE, we extend the UWE Profile and add stereotypes and tags. For example, UWE's navigation state model can be enriched for modeling application behavior in case of panic- or under-fire mode.

The PANIC MODE is useful for tumultuous regions around the world, where users might be physically forced to sign in an online service. Thereby, the offender tries to get access to critical data, as e.g., communication in social networks or to force actions, as misinformation or money transfers. Figure 4.1 depicts how the panic mode can be modeled with UWE. Exemplarily, we extend an address book example which was described in deliverable D7.1. UWE specifies that `pwd.type` can be used in guards, so that another data set (i.e. account) is loaded, in case the user logs in with a predefined password that is associated with the panic mode. It is important that the alternative account provides reasonable but uncritical fake data, e.g., addresses the offender already knows.

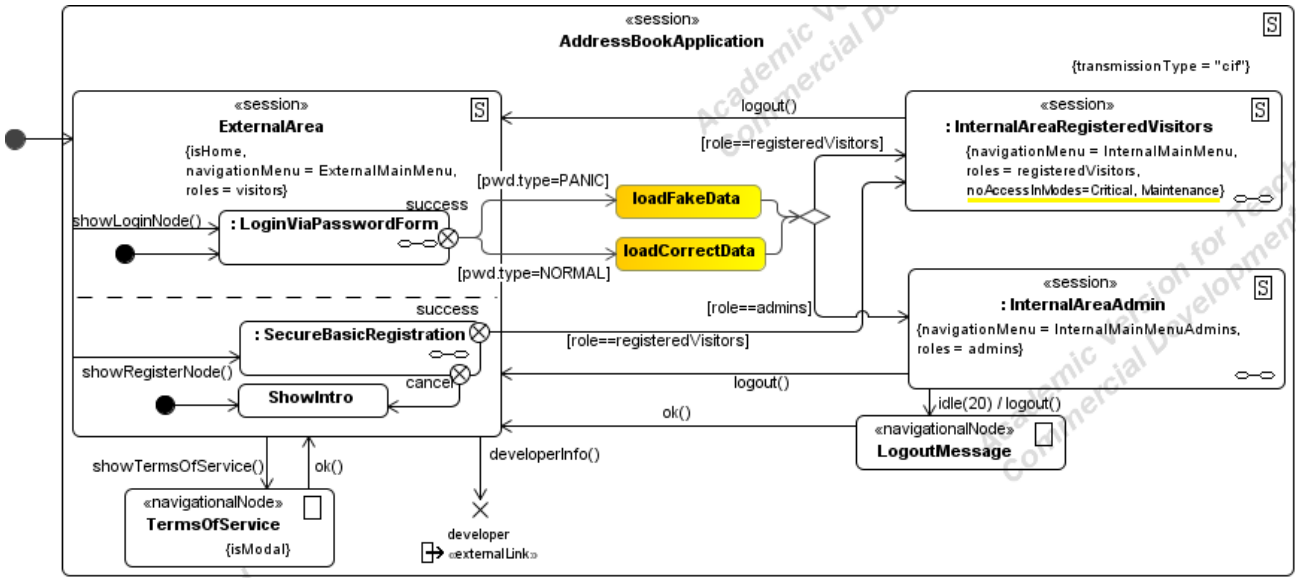

Figure 4.1: UWE extension: panic mode (example extended from D7.1)

Figure 4.1 also shows how the under-fire mode is used on the internal area for visitors. Therefore, we added the tag {noAccessInModes} to the stereotype «navigationState» which is an ancestor of the «session» stereotype. Values for the tag can be: `critical` denoting the under-fire mode, or `maintenance` to express that this functionality should not be available when server software is updated. Further values can be added individually so that, e.g., different kinds of attacks can be denoted.

Additionally, not only navigation states can be protected, but also data elements, as database entries. This is done similarly by adding a {noAccessInModes} tag to the CRUD (create, read, update, delete) access control dependencies of the Basic Rights Model. More information about the UWE models can be found at [28].

Furthermore, we add protection-specific stereotypes to UWE's context model which classify the kind of CSRF prevention that is used. Additional stereotypes specify the chosen input validation mechanism, which makes sure that user input cannot lead to attacks, as e.g., cross-site scripting (XSS)[3].

Future work includes taking further web security features into account and to validate them by applying them to modeling examples. We already started to model the customer web interface from the smart grid case study of deliverable D11.3. We plan to present the models for the case study in D11.4.

[3]XSS is a form of injection, "in which malicious scripts are injected into the otherwise benign and trusted web sites" (https://owasp.org/index.php/XSS). Injected scripts can be stored on a server and distributed to other users, or just be reflected to the local user, e.g., after submitting a script within an unprotected search form.

5 ActionGUI toolkit and applications

Purpose of the prototype	*Model-driven development of secure data-managament applications*
Modeling formalism	*DSLs for data, security, and GUI models*
Security concern	*Fine-grained access control*
Partner(s)	*IMDEA, ETH*

ActionGUI is a novel model-driven methodology for developing secure data-management applications. System developers proceed by modeling three different views of the desired application: its data model, security model, and GUI model. These models formalize respectively the application's data domain, authorization policy, and its graphical interface together with its behavior. Afterwards a model-transformation function automatically lifts the policy specified by the security model to the GUI model; this allows a separation of concerns where behavior and security are specified separately, and afterwards combined to generate a security-aware GUI model. Finally, a code generator automatically generates a multi-tier application, along with all support for access control, from the security-aware GUI model. Here we report on applications that demonstrate the usefulness of our approach and the associated ActionGUI Toolkit.

5.1 ActionGUI Toolkit

The ActionGUI Toolkit (http://www.actiongui.org) features model editors for constructing and manipulating data, security, and GUI models. Moreover, it implements our model transformation to automatically generate security-aware GUI models. Finally, it includes a code generator that, given a security-aware GUI model, automatically produces a web application, ready to be deployed, based on the following, standard three-tier architecture.

- **Presentation tier (or front-end):** Users access web applications through standard web browsers, which render the content (HTML and JavaScript) dynamically provided by the application server;
- **Application tier:** The toolkit generates Java Web Applications, implemented using the Vaadin framework. The applications run in a servlet container (such as Tomcat or GlassFish), process client's requests and, generate content, which is sent back to the client for rendering. They may also manipulate data stored in the data tier. When processing client requests, the generated application *interprets* its underlying security-aware GUI model;
- **Data tier (or back-end):** The generated application manages information stored in database servers. For each application, the toolkit generates the corresponding database schema from the application's data model.

Custom code Web applications may contain custom code for performing specific tasks such as printing files, sending messages, or exporting data in various formats. Within the ActionGUI Toolkit, custom code is declared using methods in the application's data model. The application's security model can be used to declare access control policies for executing these methods, and the GUI model can declare events that will trigger their execution. As expected, the ActionGUI Toolkit does not generate code for these methods. Instead, it includes their implementation in the generated application and when the application needs to interpret one of these methods, it simply calls the provided method.

5.2 Applications

To provide evidence of the usefulness of our approach, we report here on four web applications that we developed using ActionGUI.

- **Customer Relationship Management (CRMApp)** We have developed a web application for managing customers of a Hospital and Care Center. This application allows marketing and public relations personnel to manage contact information, including filtering contacts based on different criteria

and exporting the results in Excel files. As customer data is highly sensitive, data access is subject to a restrictive access-control policy. For example, a marketing and PR staff member can only access the contact information of those contacts previously selected as targets of a marketing campaign to which he is assigned. The application also allows a General Manager to create marketing campaigns, select the targeted patients, and assign to the campaigns marketing and PR staff members;

- **Volunteer Management (VMApp)** We have also developed a web application for managing a Care Center's Volunteer Program. Using this application, the program's coordinators can take actions such as: introduce new volunteers; create, edit, and modify tasks; and propose these tasks to the volunteers, based on the volunteers' time availability and preferences. The access-control policy stipulates, for example, that volunteers are only authorized to edit their own personal information, such as their preferences and time availability, and to accept or reject their own tasks;
- **Meal Service Management (MSMApp)** This is a web application for managing a student residence's meal service. Using this application, a resident can notify the administration whether he will have a meal at the residence's cafeteria, in which of the available time slots, and if he will bring a guest. A resident shall only edit his own meal selection and within a specific time window, which depends on the selected meal. Members of the administration can create new resident accounts, and automatically list the requested meals for each available time slot;
- **EHealth Record Management (eHRMApp)** This is a web application for managing eHealth records. It allows users with the appropriate roles to do the following: register new patients in a hospital and assign to them clinicians (doctor, nurses, etc.); retrieve patient information; register new nurses and doctors in a hospital and assign them to a ward; change nurses or doctors from one ward to another; and move patients to a different practice. The access-control policy regulates, in particular, access to the patients' highly sensitive records. These records shall only be retrieved by their handling doctors, although this policy can be relaxed in an emergency situation.

CRMApp, VMApp, and MSMApp are *commercial* applications. They were developed for real customers, and they are currently being used by their different stakeholders. In contrast, EHRMApp was developed as a case study proposed by our industrial partners in NESSoS.

We conclude this section with several considerations. As these are just based on our experience, their significance should not be overemphasized. Nevertheless they can give an impression of what it is like to use ActionGUI.

First, the learning curve for ActionGUI appears moderate. The time required to learn to model a secure data-management application depends on the modeler's familiarity with class diagrams, security diagrams and, above all, OCL. Assuming modest knowledge of UML, learning to model (non-toy) applications using ActionGUI takes less than 4 hours, which is arguably much less than what is required to learn enough of web technologies to be able to program (and correctly deploy) these applications. This estimate is based in part on experience teaching model-driven development of secure data-management applications to students at ETH Zurich and having them carry out projects using ActionGUI.

Second, the time needed to model a secure data-management application depends on the experience of the modeler (in particular, his command of OCL) and the complexity of the application (in particular, the number and the size of the OCL expressions used in the model). However, as a rule of thumb, modeling a *menu* window that contains 6 buttons that, when clicked upon, will open other windows, may take less than 30 minutes. In contrast, modeling a window that contains an *online form* with 10 text fields and a button that, when clicked upon, will first check that the entries are correctly filled and then will submit the form (i.e., update the database), may take one to two hours. And approximately the same time is required to model a *select&display* window that contains a table with 5 columns and a combo box, and that will change the information displayed in the table depending on the combo box selection. The take-home message is that modeling time is directly proportional to the number and size of the OCL expressions used in the models. This should not come as a surprise since these expressions are the ones that define the application's logic.

Finally, the main advantages of our approach concern the quality and ease of maintenance of the generated applications, which results from using our many-models-to-models transformation and our code-generator. First, this transformation effectively frees the developer from having to program fine-grained

authorization constraints and insert them at all the required places within the application's code and with the correct arguments. Except for small applications, this is a cumbersome and error-prone task, since the number of data actions associated to events may easily be on the order of several hundred. Moreover, these data actions are typically called with different arguments each time. Second, our approach supports modularity and separation of concerns. In particular, the security model can be changed independently of the GUI model, without worrying about re-programming and re-inserting all the new fine-grained authorization constraints since this is automatically done by our transformation. Again, given the large number of these checks together with the complexity of the corresponding authorizations, we argue that, for non-toy security data-management applications, our approach effectively reduces the maintenance costs.

5.3 Conclusions

Our ActionGUI methodology is supported by the ActionGUI Toolkit. Applications like those described here show the toolkit's potential for developing real-world applications. Nevertheless, there is still much work ahead to turn this toolkit into a full, robust, industrial-strength development platform. In the short term, we plan to develop improved model editors and better support for integrating custom code. In the long term, we would like to support GUIs running on different platforms, like mobile devices. We also plan to add support for handling *privacy policies*: modeling and generating code to enforce that data usage must follow the purpose for which the data was collected and may entail obligations.

6 Validation of ATL declarative transformations using transformation models and model finders

Purpose of the prototype	*Validating declarative ATL transformations using transformation models and model finders*
Modeling formalism	*MOF compliant metamodels, OCL, ATL*
Security concern	*Preservation of security properties captured as relations between model elements or written as OCL pre- and postconditions through model transformations*
Partner(s)	*ATOS, INRIA*

6.1 Background

In model-driven engineering, models constitute pivotal elements of the software to be built. If models are specified well, transformations can be employed for different purposes, e.g., to produce final code. However, it is important that models produced by a transformation from valid input models are valid, too, where validity refers to the metamodel constraints, often written in OCL. Transformation models are a way to describe this Hoare-style notion of partial correctness of model transformations using only metamodels and constraints. In this chapter we report on an automatic translation of declarative, rule-based model transformations implemented with the ATL language [1] into such transformation models, that provides an intuitive and versatile encoding of ATL into OCL that can be used for the analysis of various properties of transformations. We furthermore show how existing model satisfiability checkers for OCL-annotated metamodels can be applied for the verification of the translated ATL transformations, providing evidence for the effectiveness of our approach in practice.

6.2 Problem statement

For this deliverable the problem statement is the same as the one that we already reported for deliverable D7.3. Let us recall our setting next. A developer who is designing a model transformation typically asks the following question several times during the designing process: Do the constraints imposed on the source model plus the transformation specification guarantee that the constraints imposed on the target model hold? If the answer to this question is 'yes' for certain properties, we would say that the transformation which is being designed is correct with respect to the given sets of pre and postconditions. Namely, in our view, a model transformation is correct if and only if executing it using a constrained-valid input model as argument always results on a constrained-valid output model, where a constrained-valid input model is a model that satisfies the model transformation's preconditions and a constrained-valid output model is a model that satisfies the model transformation's postconditions.

We present here an extension to the prototype that we introduced in D7.3. This extension is based on a novel methodology that can be used to perform automatic, bounded verification of ATL [24] transformations. ATL (ATL Transformation Language) is a model transformation language and toolkit (http://www.eclipse.org/atl/). This is a research result of the collaboration between ATOS and INRIA that was published at the beginning of the third year of NESSoS [7]. This work is focused, as the work we reported last year, on checking partial correctness of declarative, rule-based transformations between metamodels with constraints. However, the solution we presnet here supplements the one presented in D7.3. More specifically, we consider the ATL transformation language and MOF [38] style metamodels that employ OCL constraints to precisely describe their domain. Nowadays, both formalisms count with sophisticated tool support and OCL is employed in almost all OMG specifications.

[1] http://www.eclipse.org/atl/

The increasingly popularity of MDE has led to a growing complexity in both models and transformations, and it is essential that transformations are correct if they are to play their key role. Otherwise, errors introduced by transformations will be propagated and may produce more errors in the subsequent MDE steps.

Model transformations can be considered as programs that operate on instances of metamodels. In this sense, we can also apply the classical notion of correctness to model transformations. More specifically, we consider the transformation language ATL [24] and metamodels in MOF [38] style (e.g., EMF [41], KM3 [25]) that employ OCL [37, 47] constraints to precisely describe their domain.

6.3 Approach

In this chapter we present a verification approach based on *transformation models*. Transformation models are a specific kind of what is commonly called a 'trace model'. Given an ATL transformation $T : \mathcal{M}_I \rightarrow \mathcal{M}_F$ from a source metamodel $\mathcal{M}_I$ to a target metamodel $\mathcal{M}_F$, a transformation model $\mathcal{M}_T$ is a metamodel that includes $\mathcal{M}_I$ and $\mathcal{M}_F$, and additional structural modeling elements and constraints in order to capture the execution semantics of T.

Our notion of a transformation model $\mathcal{M}_T$ of a transformation $T : \mathcal{M}_I \rightarrow \mathcal{M}_F$[2] aims to support the verification of partial correctness of T using $\mathcal{M}_T$ as an equivalent surrogate. A pair of an $\mathcal{M}_I$ instance M_I and an $\mathcal{M}_F$ instance M_F is related by T if and only if there is an instance of $\mathcal{M}_T$, valid w.r.t. to all constraints, whose $\mathcal{M}_I$ part is M_I and whose $\mathcal{M}_F$ part is M_F. [3]

Having translated an ATL transformation T into a purely structural transformation model $\mathcal{M}_T$ (i.e., a metamodel consisting of classes and their properties, and constraints), we can employ 'off-the-shelf' model finders (model satisfiability checkers) to verify partial correctness of T w.r.t. the metamodel constraints of $\mathcal{M}_F$ using $\mathcal{M}_T$.

In particular, we can check whether T might turn a valid input model M_I into an invalid output model M_F as follows: Let con_i with $1 \leq i \leq n$ denote the i-th constraints of $\mathcal{M}_F$. Let $\mathcal{M}_{\overline{F}i}$ denote a modified version of $\mathcal{M}_F$ stripped of all its constraints and having one new constraint $negcon_i$ that is the negation of con_i. Let $\mathcal{M}_{\overline{T}i}$ denote the transformation model constructed for $T : \mathcal{M}_I \rightarrow \mathcal{M}_{\overline{F}i}$. T is correct w.r.t. con_i if and only if $\mathcal{M}_{\overline{T}i}$ has no instance. If such an instance exist, its $\mathcal{M}_I$ is a counter example for which T produces an invalid result.

In our opinion, this approach brings advantage because it reduces the problem of verifying rule-based transformations between constrained metamodels to the problem of verifying constrained metamodels only. This way, in terms of automated verification, we can reuse existing implementations and work for model verification, benefiting from the results achieved by a broad community over a decade.

Below, we present an extension to the ATL2FOL prototype, called ATL2Alloy, that is able to automatically generate transformation models from declarative ATL transformations. Furthermore, it uses model finders that can be employed 'off-the-shelf' in practical verification for the so obtained OCL-annotated metamodels.

6.4 Prototype

In the field of MDE, ATL provides ways to produce a set of target models from a set of source models. Developed on top of the Eclipse platform, the ATL Integrated Environnement (IDE) provides a number of standard development tools (syntax highlighting, debugger, etc.) which aim to ease development of ATL transformations. The architecture of our ATL2Alloy prototype shares this toolkit with ATL2FOL as a front-end since it eases the use for the developer and because of the standard development tools that allow also the parsing and type checking of ATL transformations and OCL constraints. Thus, taking as input an ATL specification, i.e., a source and a target metamodel and an ATL transformation defined between them, our ATL2Alloy prototype's components depicted in Fig. 6.1 work as we describe next:

- The ATL transformation engine is the component where the user can specify a model-to-model ATL transformation;

[2] For typographical reasons we use $\mathcal{M}_I$ ('initial') and $\mathcal{M}_F$ ('final') to denote the input and output.

[3] In practice, we want to loosen this equivalence to hold only for those M_I for which T terminates.

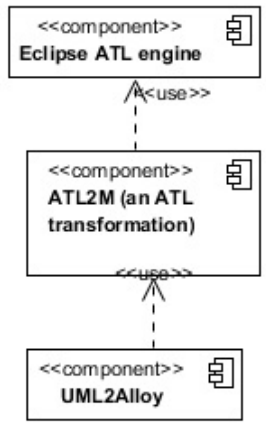

Figure 6.1: Prototype architecture

- The ATL2M component consists of a so-called 'higher-order' ATL transformation, that is, an ATL transformation that takes an ATL transformation (the one to be verified, including the input and output metamodels) that produces the corresponding transformation model following the algorithm that is specified in [7]. The metamodels and constraints are technically represented using EMF and OCLinEcore. The result of this transformation is then fed to the UML2Alloy model finder [1] to check the 'negated' transformation model (as explained above) for satisfiability;
- UML2Alloy translates the metamodel and the OCL constraints into a specification for the Alloy tool, which implements bounded verification of relational logic. In the resulting specification, each class is represented as an Alloy signature each OCL constraint is represented by exactly one Alloy fact with the same name as the OCL constraint. Thus, we can check for the constraint subsumption easily by disabling and negating the facts (one after another) for the $\mathcal{M}_F$ constraints.

Notice the benefits of the counterexamples that our method produces: The counterexamples represent at the same time the offending input model (that reveals the problem) and an explanation of the transformation execution (how the rules turn the input model into an invalid output model). In our view, this makes our method an intuitive and powerful tool for transformation developers.

While working with our prototype we got some insights into the scalability of the verification method that we would like to briefly mention here. Depending on the constraint, the verification time starts to become significant above 100 objects. We could confirm that larger class diagrams / larger instance sets do not necessarily increase the solving times, whereas harder (more overlapping, less tractable) constraints do. In this sense, we are confident that our method is applicable to at least medium size metamodels as well. However, for the verification of industrial size metamodels and transformations, we expect that further heuristics and separation of concerns strategies will be required (e.g., metamodel pruning [40]).

Example

Metamodels. Let us explain how ATL2Alloy works using an example similar to the one used to explain ATL2FOL in D7.3. We think that this will help us to show how ATL2Alloy supplements the ATL2FOL prototype.

Figure 6.2 depicts the `ER` and `REL` metamodels that are the source and target metamodels for the `ER2REL` transformation, which is depicted in Fig. 6.3. Next, we will recall these metamodels and the meaning of the ATL transformation that we use as example. In the `ER` metamodel, a schema may have entities and relationships (relships), both may contain attributes, and attributes may be keys; in the `REL` metamodel, a schema may have relations, which may have again attributes.[4] We only provide here

[4]For simplicity, we refer to the metamodel elements as schemas, entities, relationships, etc., instead of using schema type, entity

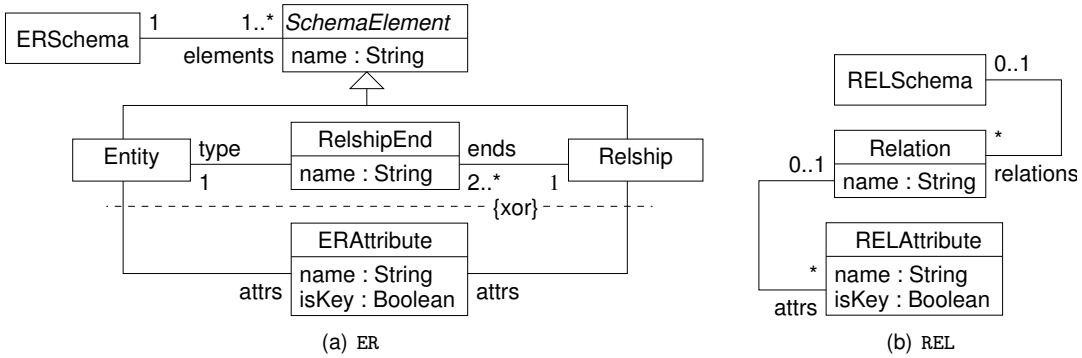

Figure 6.2: ER and REL metamodels

```
module ER2REL; create OUT : REL from IN : ER;

rule S2S { from s : ER!ERSchema to t : REL!RELSchema (name <- s.name) }

rule E2R { from s : ER!Entity
           to t : REL!Relation (name<-s.name, schema<-s.schema) }

rule R2R { from s : ER!Relship
           to   t : REL!Relation (name <-s.name, schema<-s.schema) }

rule EA2A { from att : ER!ERAttribute, ent : ER!Entity (att.entity=ent)
            to   t   : REL!RELAttribute
                       (name<-att.name, isKey<-att.isKey, relation<-ent ) }

rule RA2A { from att : ER!ERAttribute, rs  : ER!Relship (att.relship=rs)
            to   t   : REL!RELAttribute
                       (name<-att.name, isKey<-att.isKey, relation<-rs) }

rule RA2AK { from att : ER!ERAttribute, rse : ER!RelshipEnd
                        (att.entity=rse.entity and att.isKey=true)
             to   t   : REL!RELAttribute
                        (name<-att.name, isKey<-att.isKey, relation<-rse.relship)}
```

Figure 6.3: The ATL transformation ER2REL

an informal description of ER2REL. In a nutshell, the ER2REL transformation takes an instance of the ER metamodel as input and produces an instance of the REL metamodel following the transformation in Fig. 6.3. This transformation is described by *matched rules*, which are the workhorse of ATL. Matched rules define a pattern of input types and possibly a filter expression (the from-clause). Each rule is applied to each matching set of objects in the input model to create the objects in the target model that are described in the to-clause, assigning values to their properties (typically) based on the input objects' properties. The first rule in Fig. 6.3, S2S, maps ER schemas to REL schemas, the second rule E2R maps each entity to a relation, and the third rule R2R maps each relationship to a relation. The remaining three rules generate attributes for the relations. Both, entity and relationship attributes are mapped to relation attributes (rules EA2A and RA2A). Furthermore, the key attributes of the participating entities are mapped to relation attributes as well (rule RA2AK). Notice that in the property assignment, a so-called *implicit resolution* step is needed to resolve source objects to target objects: For example the binding schema<-s.schema in E2R and R2R 'silently' replaces the ERschema value of s.schema by the RELSchema object that is created for s.schema by S2S.

In Fig. 6.4, we provide a very simple extension to the ER and REL metamodels in order to illustrate the application of our prototype to security concerns. In Fig. 6.4, the ER and the REL metamodels are extended with two additional metaclasses, i.e., one to model roles and the other to model permissions for a role-based access control policy that may be required on the ER and REL elements. RoI and permission metaclasses are associated to each other and, also, permissions are associated to schemas indicating that they regulate the access to these type of resources.

type, etc..

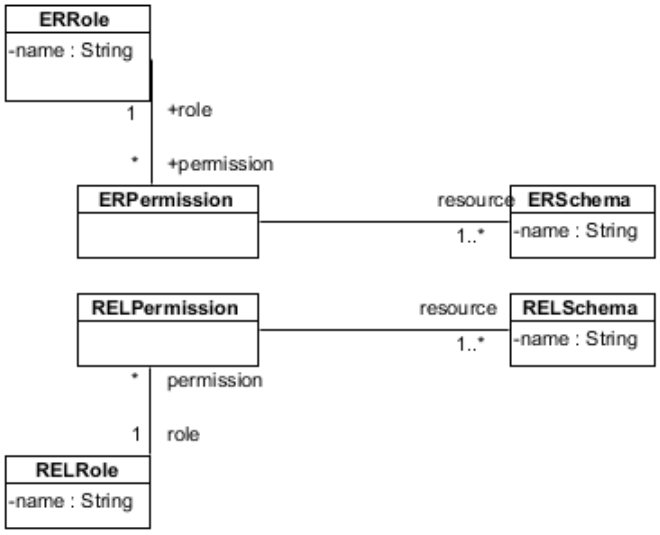

Figure 6.4: RBAC extension for the metaclass schema

In this setting, we may wonder if the security policy imposed on ER schemas is adequately translated by the transformation ER2RELSec (depicted in Fig. 6.5) to schemas in the target REL metamodel. For instance, let us take as a security precondition that *every ERSchema can be accessed by the role 'Operator'.* Is it the case that upon the execution of the transformation ER2RELSec it is true that *every RELSchema can be accessed also by a corresponding role 'Operator'?.* Notice that the ER2RELSec just contains two additional rules that extend the transformation ER2REL in order to map roles and permissions, its relations and assignments to resources. In our modeling language, the requirement that operators are always granted access to ER schemas is modeled for each schema by the existence of an instance of the metaclass Role named 'Operator', an instance of the metaclass permission and a link between this role and permission instances, and another link between such permission and any instance of the class ERSchema.[5] The OCL expression capturing this security precondition is:

```
inv secpre1:
ERSchema.allInstances()->forall(x | ERRole.allInstances()->exists(r | r.name =
    'Operator' and ERPermission.allInstances()->exists(p | r.permission->includes(p)
    and p.resource->includes(x))))
```

Similarly, the OCL expression capturing the intended security postcondition is:

```
inv secpost1:
RELSchema.allInstances()->forall(x | RELRole.allInstances()->exists(r | r.name =
    'Operator' and RELPermission.allInstances()->exists(p |
    r.relpermission->includes(r) and p.relresource->includes(x))))
```

Let us next formalize this problem according to our methodology. Let us consider the transformation $ER2REL^*$: $ER \rightarrow REL^*$, where REL^* is the metamodel REL but having only the invariant `not(secpost1)` as its constraint. Let then $\mathcal{M}_{ER2REL^*}$ be a transformation model for $ER2REL^*$. Then, $ER2REL^*$ is correct w.r.t. `secpost1` if and only if $\mathcal{M}_{ER2REL^*}$ has no instance. If such an instance exist, its part in ER is a counter example for which $ER2REL$ produces an invalid result.

Once the security pre and postcondition and the rules depicted in Figure 6.5 are mapped to Alloy using our prototype that implements the transformation algorithm that we describe in [7], we can automatically search for instances of $\mathcal{M}_{ER2REL^*}$. Naturally, while we are incrementing the number of objects allowed to build such instance our confidence grows on our transformation to be correct since such instance is not found by Alloy. However, we cannot derive a definitive conclusion since with this method we are only performing bounded checkings. But, since we now really suspect our transformation to be correct, we employ the part ATL2FOL of our prototype using as input the transformation $ER2REL^*$. We find that Z3 returns 'unsat' showing that our transformation is definitely correct w.r.t. the secpre1 and secpost1 (the

[5]Notice that at this step, for simplicity reasons, we avoid talking about which type of access, i.e., which kind of actions (write, read, etc.) can a role perform on schemas.

```
module ER2RELSec;
create OUT : REL from IN : ER;

rule Rol2Rol {from r : ER!ERRole
              to k : REL!RELRole (name <- r.name, permission <-r.permission)}

rule Perm2Perm {from p : ER!ERPermission
                to q : REL!RELPermission (name <- p.name, resource <-p.resource)}

rule S2S { from s : ER!ERSchema to t : REL!RELSchema (name <- s.name) }

rule E2R { from s : ER!Entity
           to t : REL!Relation (name<-s.name, schema<-s.schema) }

rule R2R { from s : ER!Relship
           to   t : REL!Relation (name <-s.name, schema<-s.schema) }

rule EA2A { from att : ER!ERAttribute, ent : ER!Entity (att.entity=ent)
            to   t   : REL!RELAttribute
                       (name<-att.name, isKey<-att.isKey, relation<-ent ) }

rule RA2A { from att : ER!ERAttribute, rs  : ER!Relship (att.relship=rs)
            to   t   : REL!RELAttribute
                       (name<-att.name, isKey<-att.isKey, relation<-rs) }

rule RA2AK { from att : ER!ERAttribute, rse : ER!RelshipEnd
                        (att.entity=rse.entity and att.isKey=true)
             to   t   : REL!RELAttribute
                        (name<-att.name, isKey<-att.isKey, relation<-rse.relship)}
```

Figure 6.5: Extension of ER2REL to map RBAC structures

interested reader can find details of how the example is processed by ATL2FOL in deliverable D7.3).

Consider now the same setting for our example keeping as precondition `secpre1` while considering as postcondition only `secpost2`:

```
inv secpost2:
RELSchema.allInstances->forall(x| relrole.allinstances->forall(r|r.name='Operator'
    implies RELpermission.allInstances->exists(p|r.relpermission->includes(r) and
    p.relresource->includes(x))))
```

Following the same methodology as for the previous example, we feed to ATL2Alloy the transformation that has as its final metamodel `REL` with the only constraint `not(secpost2)`. For this example, ATL2Alloy finds the instance depicted in Figure 6.6 that shows that the transformation `ER2REL*` is not correct with respect to `secpre1` and `secpost2` since it produces a bad instance of the final metamodel (i.e., an instance that do not fulfil `secpost2` from a good instance of the initial metamodel (i.e., an instance that does fulfil `secpre1`).

We want to emphasize that the verification process can be automated as a “black box” technology, in the sense that the transformation developer is in contact only with models, in which the generated transformation models and their instances have a familiar representation for him. In Figure 6.7 we show the prototypes described in D7.3 and this document shares the modeling interface so they allow the user to perform for the same transformation both bounded and unbounded verification.

In the future, we plan to explore the capabilities of different model finders as backends to our approach, in order to evaluate which are best suited for this kind of verification. Regarding ATL, we have already implemented an important subset of ATL, but we will incorporate (a restricted form) of so called *lazy rules*, which can be found in several transformations. Last but not least, comprehensive case studies must give more feedback on the applicability of our work.

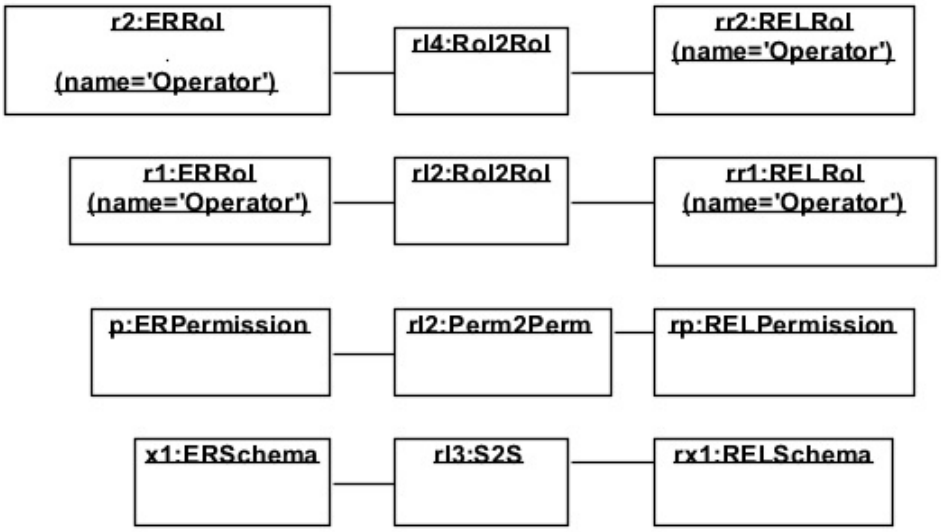

Figure 6.6: Instance of the counter example found by Alloy

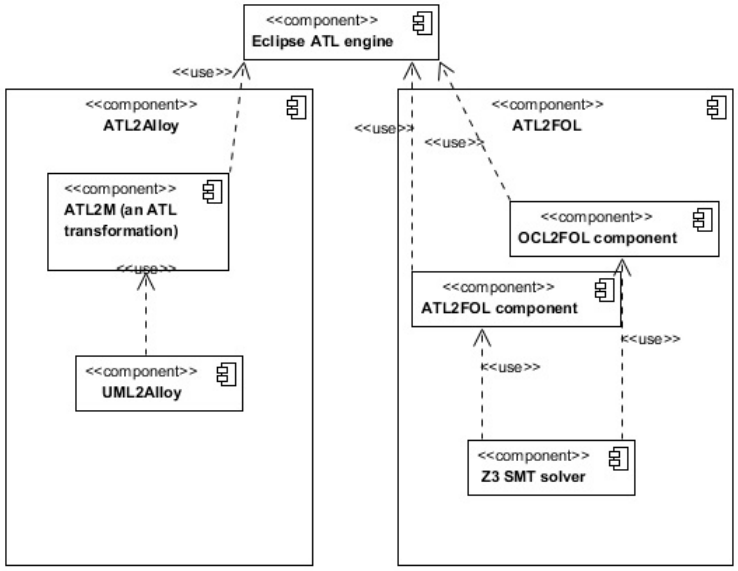

Figure 6.7: Architecture of the prototype supporting ATL verification

7 Integrating intrusion detection in a models@runtime framework

Purpose of the prototype	*Secure runtime reconfiguration*
Modeling formalism	*Kevoree*
Security concern	*Intrusion detection*
Partner(s)	*INRIA*

Future Internet systems such as cloud platforms or pervasive systems (e.g. sensor networks, peer-to-peer applications on mobile personal devices, etc) are inherently dynamically adaptive in the sense that they are able to change their architecture while they are deployed and running. Changes of architecture occur mostly without human intervention, as a result of adaptation policies executed by the system itself or by an external supervision system. Needs for adaptation come either from changes in the needs of the dynamic system (for instance to support new services or implementations), or from changes in the system's environment (for instance to take into account changes in resource availability or cost, load peaks, etc.). Adaptation can be organized using the MAPE (Monitor, Analyze, Plan, Execute) paradigm [20].

Software design techniques such as Models@runtime [5] are a powerful way of building self-managing systems, leveraging continuous design and eternal system management. Models@runtime relies on the continuous management of architecture models [31] that abstract platform details and describe the current architectures and candidates for architecture changes. Prior to a configuration change, the analyze part of the MAPE loop builds a set of models describing potential new architectures of the system. This analysis aims at checking that the planned system evolution stays within limits and constraints defined by the human designer.

7.1 Security issues for dynamically adaptive systems

While all systems are targets for attacks, dynamic architectures bring in new security-related issues. Because of their dynamic nature, they offer many possible configurations, and hence increase the system's surface for attack. Because of their intrinsic mechanisms for the application of architecture changes, they ease the task of attackers: a malicious architecture model could be propagated effortlessly and compromise the whole system. Because of their autonomy and self-care principles, dynamic systems must detect and counter attacks autonomously, without the help of a human security expert.

Dynamic architectures must therefore provide self-protection, including architecture-based self-protection [48]. Security must be integrated in the MAPE loop, with appropriate means for attack detection, selection of countermeasures and application of countermeasures using Models@runtime.

7.1.1 System structure and threat model

We have developed a dynamic component-based framework [10], which implements Models@runtime for heterogeneous distributed execution platforms. Kevoree simplifies the designer's work by offering advanced dynamic model management and taking care of complex issues such as distributed communication, distributed data consistency and dynamic architecture deployment. To tackle security problems such as the ones mentioned in the previous section, we are currently integrating a set of concepts and implementations in our Kevoree framework to support security policies.

The main concepts in Kevoree are as follows.

- nodes: containers for execution of components and manager of the models at runtime implementation;
- channels: distributed communication entities that interconnects two components or more, and can be 1-1, 1-N, N-N, with a variety of communication semantics (e.g. 1-1 sequential lossless, 1-N causal broadcast, etc);

- groups: distributed data consistency managers, with a variety of data consistency semantics.

The system boundary in terms of security is the set of components defined in the currently deployed configuration. This system has the following subsystems:

- a set of local operating systems on each physical execution node (e.g. Arduino loader, Android, Linux, etc), responsible for VM execution and network communication;
- a set of Kevoree nodes, responsible for Models@runtime implementation (keeping the architectural model in bidirectional synchronization with the concrete execution platform);
- on each Kevoree node: a set of Kevoree components for application, a set of Kevoree channels and a set of group stubs;
- there is at least one master node, which is used to bootstrap the deployment of the system, disseminate architectural constraints when they change, and setup a distributed key infrastructure in the system.

We are currently considering the following general types of threat: network attacks from outside the system (i.e. not Kevoree specific), tampering with models during their propagation in the set of nodes, and tampering with the distributed algorithm for analysis and planning. We focus especially on the last two types, because they are specific to weaknesses of a models at runtime implementation of continuous design.

The main security policy rules are that:

- policies must be enforced automatically, there will be no human being behind a console to react;
- architecture changes proposed by nodes are evaluated by the set of nodes and accepted or rejected by a poll among the nodes;
- architectural changes proposed by a master node are accepted without poll if they are signed with a proper key.

7.2 General approach for securing an adaptive environment

Our current approach to secure the Kevoree framework relies on an intrusion detection system to detect suspicious entities in a distributed system. We are currently focusing on securing a cloud platform. We selected cloud platforms since the elastic management of resources makes them very dynamic, but also challenges most current solutions for intrusion detection and autonomous protection. Our solution (figure 7.1) relies on the following design principles:

- intrusion detection is done at a low communication level (OS level), by running a monitoring tool on a subset of the nodes;
- communication and behavior patterns observed on a node are analyzed at the node level to detect insider attacks (defense in depth);
- models are signed using a public distributed key infrastructure;
- automated reaction to intrusion by application of a strategy to counter the intrusion (Analyze, Plan);
- secure private polling to implement the automated reaction using a distributed analysis and decision system among the components of the system.

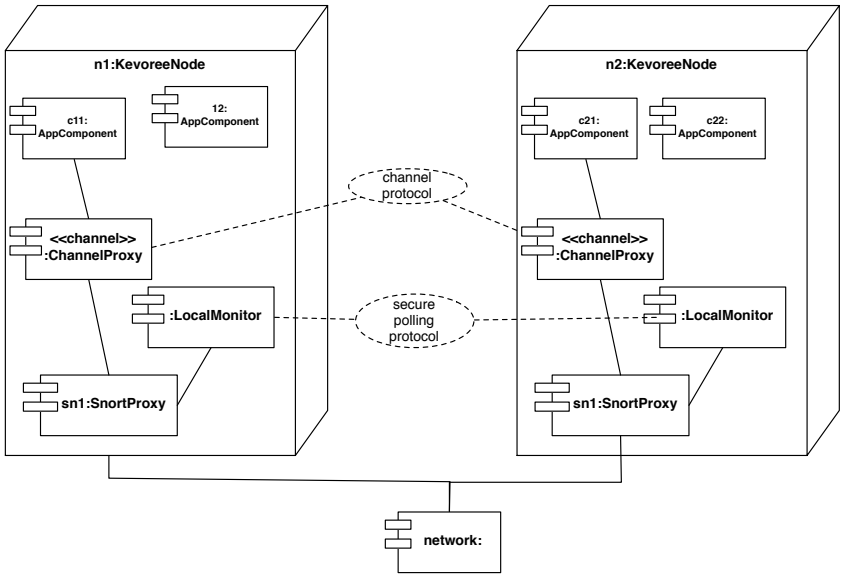

Figure 7.1: A secure architecture for adaptive FI system.

7.2.1 Intrusion detection

We use a gradual response mechanism to control the overhead associated with monitoring and countermeasures application, aiming at a reasonable percentage of false positives. The first layer of monitoring operates at a network frame level. In its cheapest configuration, a Snort-based monitor node observes all local network traffic and therefore can detect attacks that occur on nodes on the same local network than the monitor. In the general case, there will be at least a monitor node on each local network, up to one monitor on each node. The set of monitors are connected through one or more secure Kevoree channels. The Snort rule database is setup and updated by security experts, and disseminated in the architecture using the secure Kevoree model dissemination mechanisms.

7.2.2 Model based countermeasures

Once a monitor detects a suspicious network activity, it starts an analysis phase. This phase produces one or more new architecture models, which should counter the attack.

The construction of a new model (or a set of new models) uses model transformations described by patterns. More precisely, when one intrusion detection rule matches then one or more new models are produced from the existing architecture model by executing a sequence of elementary transformations. These transformations are setup by a security and a design experts. These transformations need to consider a trade-off between multiple objectives (ensure security in the system, keep a good quality for the application, handle the load, etc.). We address this through automatic multi-objective search, using genetic algorithms that explore the space of possible models that can address the threat while satisfying requirements for quality and functionality. We leverage our recent work on for model synthesis [9] to perform self-adaptation of countermeasure patterns.

7.2.3 Secure private polling

The new models are generated by the monitor that detected an intrusion. However, as the dissemination of a new architecture has an impact on several nodes, the new model must be proposed to other monitors. The other monitors must then vote to accept the new model or not. As the proposing monitor may be compromised, and as the voters may also be compromised, we use a secure private polling distributed algorithm. We are currently investigating the SPP algorithm by Gambs et al. [12] to perform this poll.

The algorithm withstands up to n(1/2 - epsilon) compromised nodes. We are also investigating whether the use of the Models@runtime paradigm can lead to a more efficient SPP variant, especially for a better scalability.

8 Trust-based runtime reconfiguration for self-adaptive systems

Purpose of the prototype	*Building trust and reputation models in services and applications*
Modeling formalism	*Java, Kevoree*
Security concern	*Trust and reputation*
Partner(s)	*UMA, INRIA*

The Future Internet (FI) scenarios are bringing two important changes in the ICT world. On the one hand, the uprising of the service-oriented vision enables the on-the-fly improvement of the features offered to users. Applications become more dynamic and require rapid adaptations to meet new requirements or respond to environmental changes. On the other hand, the emergence of the Internet of Things (IoT) is bringing the seamless integration between the physical and the virtual worlds. As a consequence, both services and systems as a whole must adapt to dynamic changes in hardware, firmware and software, including the unpredictable arrival or disappearance of devices and software components.

The aforementioned reasons prevent system architects and designers to envision all possible situations an application will have to cope with, and the boundaries between design and runtime are blurring [15]. This calls for new software engineering approaches that allow keeping an abstract representation of a running system in order to reason about changes and drive dynamic reconfiguration, leading to the so-called 'models@runtime' paradigm [5].

Security is a crucial issue that must be addressed in order to guarantee the successful deployments of FI scenarios [46]. Increasing security in FI applications entails that trust relationships between components, applications and system environments cannot be taken for granted any more, and must be explicitly declared, monitored and changed accordingly. In fact, we argue that the management of these trust relationships, together with the notion of reputation, can drive the reasoning process required in self-adaptive systems.

The goal that we pursue is to provide developers with a development framework to build trust-aware and self-adaptive applications. This way, self-adaptability can also be guided by the trust relationships and entities' reputation, increasing the security of the developed applications.

8.1 Kevoree

Models@runtime [5] refers to model-driven approaches that aim to tame the complexity of software and system dynamic adaptation, pushing the idea of reflection one step further. Kevoree [10] is an open-source dynamic component model that relies on models at runtime to properly support the design and dynamic adaptation of distributed systems.

Seven concepts constitute the basis of the Kevoree component metamodel, as shown in Figure 8.1. A *node* models a device on which software *components* can be deployed, whereas a *group* defines a set of nodes that share the same representation of the reflecting architectural model. A *port* represents an operation that a component provides or requires. A *binding* represents the communication between a port and a *channel*, which in turn models the semantics of communication. The core library of Kevoree implements these concepts for several platforms such as Java, Android or Arduino.

Kevoree follows the process shown in Figure 8.2 in order to implement models@runtime. When a new model, namely a target model, is generated, this target model is checked and validated to ensure a well-formed system configuration. Then it will be compared with the current model of the running system. This comparison generates a so-called adaptation model that contains a set of abstract primitives required to move from the current model to the target model. These primitives will be turned into concrete primitives that the running platform can execute. Thus, the synchrony between the system model and the running system is kept.

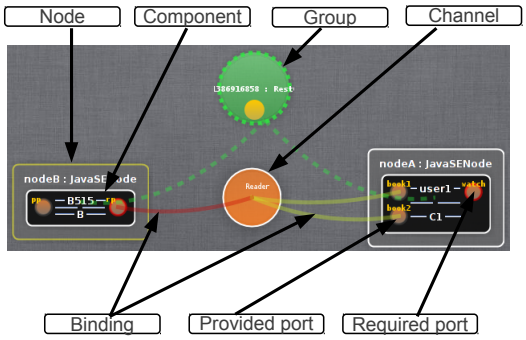

Figure 8.1: Kevoree architectural elements

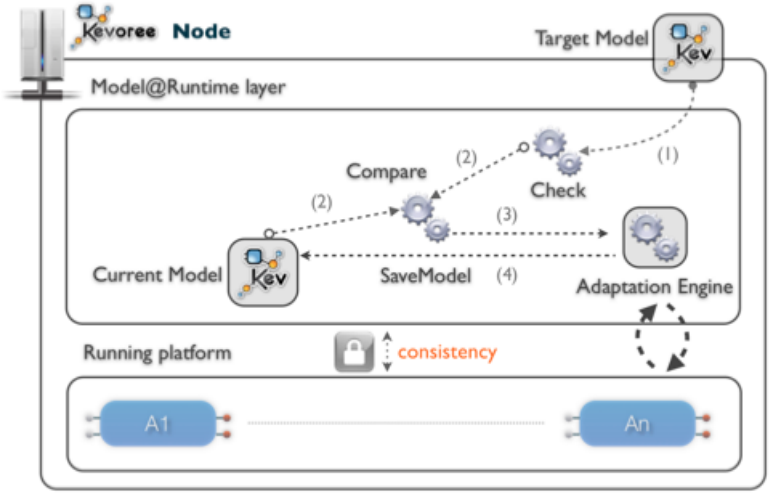

Figure 8.2: Adaptation

8.2 Approach

Up to now, the Kevoree platform does not support reasoning about qualitative or security-related concerns, and therefore any architectural element such as a node or a software component can join the system without further checks. Also, there are no criteria to guide the runtime changes. We aim at enriching these architectural elements with trust and reputation capabilities in order to support decision-making for reconfiguration.

The approach followed to achieve this is depicted in Figure 8.3, where a simple system with two communicating nodes is represented. As we explained above, both nodes share a model that represents the system (system metamodel). We add now a trust model that is also shared by both nodes. This model is an instantiation of a trust metamodel that captures knowledge of trust and reputation models as explained by Moyano et al. [36, 33, 34].

In order to provide developers with a framework to implement trust-aware self-adaptive systems, we offer two APIs. The trust API allows instatiating the trust metamodel, creating a trust model among nodes and components. For example, using this API the developer can specify which node acts as a trustor and which as a trustee or the engine used to update the trust relationships.

On the other hand, the reconfiguration API allows developers to specify rules for changing the system as a response to changes in trust. For example, developers could specify that when trust falls below a certain threshold, the binding among the nodes or components is removed.

The code in Figure 8.4 shows how a developer can create a trustor component. The yellow box depicts a dictionary, that is, a set of pairs *key, value*. These values can be changed at runtime, for instance, using

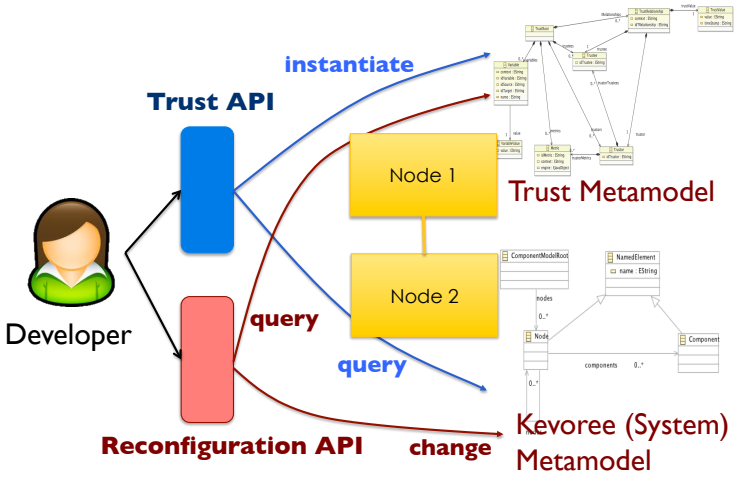

Figure 8.3: Approach

the Kevoree editor or kevscript instructions. For a trustor, we define a name for the context under which its trust relationships will take place, and the engine used to update its trust relationships.

Then, we specify that the trustor is a Kevoree component by annotating it with *@ComponentType* and extending the *AbstractComponentType* abstract class. The method *start* is executed when a component is first initialized. We specify then that any trustor component should first initialize its trust relationships.

```
@DictionaryType({
        @DictionaryAttribute(name = "trustContext" , defaultValue
= "myContext",optional = false),
        @DictionaryAttribute(name = "metric", defaultValue
="MyTrustEngine", optional = false)
}    )
                                                    Change at Runtime
@ComponentType
public class Trustor extends AbstractComponentType {

        public void start() {
                …
                initializeTrustRelationships();
        }
}
```

Figure 8.4: Creating a trustor component

Figure 8.5 shows part of the implementation of *initializeTrustRelationships*. It basically retrieves all the components flagged as *trustees* that share the same *context* as the trustor. Then, the method *addTrustRelationship* will be in charge of interacting with the trust metamodel in order to instantiate it appropriately.

Up to now, we have described the implementation of the API that will be provided to developers. When they want to use this API, they would simply need to create a component that extends from the *trustor* component created in Figure 8.4. By implementing an interface for *metric computation*, the developer will be able to specify the engine used to update the trustor's trust relationships.

Reconfiguration rules can then be written by developers in order to specify changes at runtime as a response to changes in trust. For this purpose, developers can use Kevscript, a script language provided by Kevoree that allows changing a system at runtime, as shown in Figure 8.6.

More details about this work are provided in a paper published in the Trust Managament'13 conference proceedings [32].

```
HashMap<String, List<String>> trustees =
GetHelper.getTrusteeInstanceName(getModelServ
ice().getLastModel(), context, trusteeType);
…
```

Access to Kevoree Metamodel

```
//idTrustee is a list of trustees
for (int i = 0; i < idTrustee.size(); i++) {
   addTrustRelationship(m, context,
                idTrustor, idTrustee.get(i));
}
```

Figure 8.5: Initializing trust relationships

```
IF TRUSTOR.TRUST(TRUSTEE) < THRESHOLD
    THEN  {
//KEVSCRIPT INSTRUCTIONS
 addNode node1 : JavaSENode
 addNode node2 : JavaSENode
 addChannel  mysockChannel : SocketChannel { }
 bind fake1.textEntered@node1 => mysockChannel
 bind fake2.showText@node2 => mysockChannel
….}
```

Figure 8.6: Changing the system at runtime using Kevscript

9 Relation to other WPs

As mentioned in the introduction, the model-based solutions for secure software architecture developed in WP7 are meant to assist the transition between the expression of security requirements and the actual deployment of security mechanisms on a runtime platform. In addition, some of the model-based solutions developed here, support the formal modeling of security properties that must be checked on the system's model at runtime. These two elements explain the existence of strong interactions between some prototypes developed in this workpackage and methods and tools developed in WP6 (focusing on the expression of security requirements that must then be captured in secure models), WP8 (focusing on the development of runtime platforms on which the models developed with WP7 solutions could be deployed) and WP9 (which develops verification solutions that can be used either to verify security properties on models or to verify that the models developed with WP7 solutions are correctly implemented).

In particular, the following connections are currently established:

- Connections with WP2: the trust and reputation framework will be bundled in the SDE. The OCL2FOL (OCL to first order logic) module of ActionGUI is in the SDE;
- Connections with WP6: The work by Moyano et al. [35] proposes a UML profile for trust and reputation, where the requirements engineer can specify trust and reputation requirements and its relationships with the business logic of the application. This work could be a natural step prior to defining the architecture and implementation of a trust-aware application, which is what it is proposed in this deliverable. Using the profile in D6.4, we can define the trust requirements of a system, define the nodes and components running on these nodes, and specify the trust workings of the system. Mapping from this requirements world to the architecture and implementation world should not be difficult. The cloud system analysis pattern of Section 2 is used in WP 6 as a basis for creating a method for calculating trust values in cloud computing scenarios;
- Connections with WP8: the main relation with WP8 is in the prototype of Section 7, since this tool proposes to adapt the system at runtime, according to changes in the access control policy. This tool strongly relies on reflexive middleware and dynamically adaptive platforms;
- Connections with WP9: There is a strong interaction between the work on ActionGUI and the early verification task 9.1, because formal techniques are being developed in the latter to analyse ActionGUI models. The modular verification technique presented in section 3 is also related to task 9.1. Additionally, UWE's extension (chapter 4) as part of the transformation to a Scala DSL (called TextualUWE) and the OCL2FOL transformation are both connected to D9.4;
- Connections with WP10: the PACTS method described in Section 2 relates to WP10, because it uses the CORAS risk analysis method in step 5 of PACTS. CORAS is a core research topic in WP10.

10 Conclusion

This deliverable presents advances in WP7's development of prototype tools for secure architecture and design. Three of these prototypes result from the integration of techniques coming from several partners. All three collaborations have been established during the NESSoS project.

The prototypes presented here support the different activities included in the workpackage's tasks:

- Task 7.2 Model-based decomposition of security concerns: UWE (Section 4) and ActionGUI (Section 5) support modeling secure web applications through a clear separation between architecture and security (access control) concerns; the framework introduced in section 8 supports modeling trust and reputation for further integration in service-oriented architectures;
- Task 7.3 Composition and adaption of security concerns: the prototype tools presented in Section 8 and Section 7 support the dynamic reconfiguration of secure software architectures;
- Task 7.4 Reusable architectural know-how: the prototype of Section 2 supports reuse through well-defined patterns for the design of secure cloud infrastructures; Section 3 introduces an original technique for the efficient verification of security properties over reusable security model-based patterns.

These prototypes embed initial concrete solutions to address the challenges that had been identified at the end of year one:

- UML-based model-driven security has been strengthen through the efforts from LMU, IMDEA and ETH. The advances in UWE and in ActionGUI offer significant contributions to the state of the art of model-based security engineering;
- The prototype tools presented in Section 8 and Section 7 are encouraging prototypes that show the feasibility of runtime adaptation mechanisms for RBAC policies. These prototype tool chains are founded on advanced knowledge about dynamic adaptation in this domain (impact on requirements and runtime system). However, more experiments are needed to evaluate the benefits of these integrated techniques;
- Activities on reusable architecture know-how started only in year 2, leading to two prototypes. In particular, the deployment of security architecture patterns for cloud applications will be a major challenge for the coming period.

11 NESSoS WP 7 Third-year Publications

1. F. Moyano, B. Baudry, and J. Lopez. Towards trust-aware and self-adaptive systems. In *Trust Management VII*, pages 255–262. Springer, 2013.

2. F. Moyano, C. Fernandez-Gago, and J. Lopez. A Trust and Reputation Framework. In *Doctoral Symposium of the International Symposium on Engineering Secure Software and Systems (ESSoS-DS 2013)*, pages 7–12. CEUR-WS, 2013.

3. F. Moyano, C. Fernandez-Gago, and J. Lopez. Towards Engineering Trust-aware Future Internet Systems. In *3rd International Workshop on Information Systems Security Engineering (WISSE 2013)*, pages 490–501, Springer, 2013.

4. K. Beckers, I. Côté, S. Faßbender, M. Heisel, and S. Hofbauer. A pattern-based method for establishing a cloud-specific information security management system In Requirements Engineering Journal, pages 1-53, Springer, 2013.

5. F. Büttner, M. Egea, E. Guerra, and J. de Lara. Checking model transformation refinement. In *ICMT*, pages 158–173, 2013.

6. M. Egea, F. Paci, M. Petrocchi, and N. Zannone. Persona - a personalized data protection framework. In *IFIPTM*, pages 272–280, 2013.

7. Thomas Heyman. A Formal Analysis Technique for Secure Software Architectures. PhD thesis, KU Leuven, March 2013.

Bibliography

[1] K. Anastasakis, B. Bordbar, G. Georg, and I.Ray. UML2Alloy: A Challenging Model Transformation. In *MoDELS 2007*, volume 4735 of *LNCS*. Springer, 2007.

[2] D. A. Basin, M. Clavel, M. Egea, M. A. García de Dios, C. Dania, G. Ortiz, and J. Valdazo. Model-driven development of security-aware guis for data-centric applications. In A. Aldini and R. Gorrieri, editors, *Foundations of Security Analysis and Design VI - FOSAD Tutorial Lectures*, volume 6858 of *Lecture Notes in Computer Science*, pages 101–124. Springer, 2011.

[3] K. Beckers, I. Côté, S. Faßbender, M. Heisel, and S. Hofbauer. A pattern-based method for establishing a cloud-specific information security management system. *Requirements Engineering*, pages 1–53, 2013.

[4] K. Beckers, J.-C. Küster, S. Faßbender, and H. Schmidt. Pattern-based support for context establishment and asset identification of the ISO 27000 in the field of cloud computing. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*, pages 327–333. IEEE Computer Society, 2011.

[5] G. Blair, N. Bencomo, and R. France. Models@ runtime. *IEEE Computer*, 2009.

[6] M. Busch, A. Knapp, and N. Koch. Modeling Secure Navigation in Web Information Systems. In J. Grabis and M. Kirikova, editors, *10th Int. Conf. on Business Perspectives in Informatics Research*, LNBIP, pages 239–253. Springer Verlag, 2011.

[7] F. Büttner, M. Egea, J. Cabot, and M. Gogolla. Verification of atl transformations using transformation models and model finders. In *ICFEM*, pages 198–213, 2012.

[8] Cloud Security Alliance (CSA). Top threats to cloud computing v1.0, 2010. https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf.

[9] D. Efstathiou, P. McBurney, S. Zschaler, and J. Bourcier. Exploring Optimal Service Compositions in Highly Heterogeneous and Dynamic Service-Based Systems. In *SSBSE*, 2013.

[10] F. Fouquet, B. Morin, F. Fleurey, O. Barais, N. Plouzeau, and J. Jézéquel. A dynamic component model for cyber physical systems. In *CBSE*, 2012.

[11] M. Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1996.

[12] S. Gambs, R. Guerraoui, H. Harkous, F. Huc, and A.-M. Kermarrec. Scalable and secure polling in dynamic distributed networks. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 181–190. IEEE, 2012.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[14] Gartner. Assessing the security risks of cloud computing, 2008. http://www.gartner.com/id=685308.

[15] C. Ghezzi. The fading boundary between development time and run time, 2011. Invited talk for the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.

[16] T. Heyman. *A Formal Analysis Technique for Secure Software Architectures (Een formele analysetechniek voor veilige softwarearchitecturen)*. PhD thesis, KU Leuven, March 2013.

[17] T. Heyman, R. Scandariato, and W. Joosen. Security in context: analysis and refinement of software architectures. In *Annual IEEE Computer Software and Applications Conference*, July 2010.

[18] T. Heyman, R. Scandariato, and W. Joosen. Reusable formal models for secure software architectures. In *Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture and 6th European Conference on Software Architecture*. IEEE, 2012.

[19] J.-C. Hourcade, Y. Neuvo, R. Posch, R. Saracco, and W. Wahlster. Future internet 2020, visions of an industry group. Technical report, 2009.

[20] M. C. Huebscher and J. A. McCann. A survey of autonomic computingâĂŤdegrees, models, and applications. *ACM Computing Surveys (CSUR)*, 40(3):7, 2008.

[21] ISO/IEC. Information technology - Security techniques - Information security management systems - Requirements. ISO/IEC 27001, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), 2005.

[22] D. Jackson. *Software Abstractions: logic, language and analysis*. The MIT Press, 2006.

[23] M. Jackson. *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley, 2001.

[24] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008.

[25] F. Jouault and J. Bézivin. KM3: A DSL for Metamodel Specification. In *Formal Methods for Open Object-Based Distributed Systems, FMOODS 2006, Proc.*, volume 4037 of *LNCS*, pages 171–185, 2006.

[26] J. Jürjens. UMLsec: Extending UML for secure systems development. *UML – The Unified Modeling Language*, pages 1–9, 2002.

[27] N. Koch, A. Knapp, G. Zhang, and H. Baumeister. UML-based Web Engineering: An Approach based on Standards. In *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, pages 157–191. Springer, 2008.

[28] LMU. Web Engineering Group. UWE Website. http://uwe.pst.ifi.lmu.de/.

[29] S. Meliá, J. Gómez, S. Pérez, and O. Díaz. A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. In *Proc. 8th Int. Conf. Web Engineering (ICWE'08)*, pages 13–23. IEEE, 2008.

[30] N. Moreno, P. Fraternali, and A. Vallecillo. WebML modelling in UML. *IET Software*, 1(3):67, 2007.

[31] B. Morin, O. Barais, J. Jézéquel, F. Fleurey, and A. Solberg. Models@ runtime to support dynamic adaptation. *IEEE Computer*, 2009.

[32] F. Moyano, B. Baudry, and J. Lopez. Towards trust-aware and self-adaptive systems. In *Trust Management VII*, pages 255–262. Springer, 2013.

[33] F. Moyano, C. Fernandez-Gago, and J. Lopez. Building trust and reputation in: A development framework for trust models implementation. In *8th International Workshop on Security and Trust Management (STM 2012)*, Pisa, Sep 2012.

[34] F. Moyano, C. Fernandez-Gago, and J. Lopez. A Trust and Reputation Framework. In *Doctoral Symposium of the International Symposium on Engineering Secure Software and Systems (ESSoS-DS 2013)*, pages 7–12. CEUR-WS, 2013.

[35] F. Moyano, C. Fernandez-Gago, and J. Lopez. Towards engineering trust-aware future internet systems. In X. Franch and P. Soffer, editors, *3rd International Workshop on Information Systems Security Engineering (WISSE 2013)*, volume 148 of *LNBIP*, pages 490–501, Valencia, Jun 2013 2013. Springer-Verlag, Springer-Verlag.

[36] F. Moyano, C. Fernandez-Gago, and J. Lopez. A conceptual framework for trust models. In *9th International Conference on Trust, Privacy & Security in Digital Business (TrustBus 2012)*, September 2012 In Press.

[37] OMG. *The Object Constraint Language Specification v. 2.2 (Document formal/2010-02-01)*. Object Management Group, Inc., Internet: http://www.omg.org/spec/OCL/2.2/, 2010.

[38] OMG. *Meta Object Facility (MOF) Core Specification 2.4.1 (Document formal/2011-08-07)*. Object Management Group, Inc., Internet: http://www.omg.org, 2011.

[39] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns: Integrating Security and Systems Engineering*. Wiley, 2006.

[40] S. Sen, N. Moha, B. Baudry, and J.-M. Jézéquel. Meta-model Pruning. In *MODELS 2009, Proc.*, volume 5795 of *LNCS*, pages 32–46. Springer, 2009.

[41] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Longman, Amsterdam, 2nd edition, 2008.

[42] UML Revision Task Force. *OMG Unified Modeling Language (UML), Superstructure*. http://www.omg.org/spec/UML/2.3/Superstructure/PDF.

[43] UML Revision Task Force. OMG Object Constraint Language: Reference, February 2010.

[44] F. Valverde and O. Pastor. Applying Interaction Patterns: Towards a Model-Driven Approach for Rich Internet Applications Development. In *Proc. 7th Int. Wsh. Web-Oriented Software Technologies (IWWOST'08)*, 2008.

[45] A. van Lamsweerde. *Requirements Engineering: from system goals to UML models to software specifications*. Wiley, March 2009.

[46] D. van Rooy and J. Bus. Trust and privacy in the future internet - a research perspective. *Identity in the Information Society*, 3(2):397–404, Aug. 2010.

[47] J. B. Warmer and A. G. Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley, 2nd edition, 2003.

[48] E. Yuan, S. Malek, B. Schmerl, D. Garlan, and J. Gennari. Architecture-based self-protecting software systems. In *Proceedings of the Ninth International Conference on the Quality of Software Architectures (QoSA 2013)*, 2013.